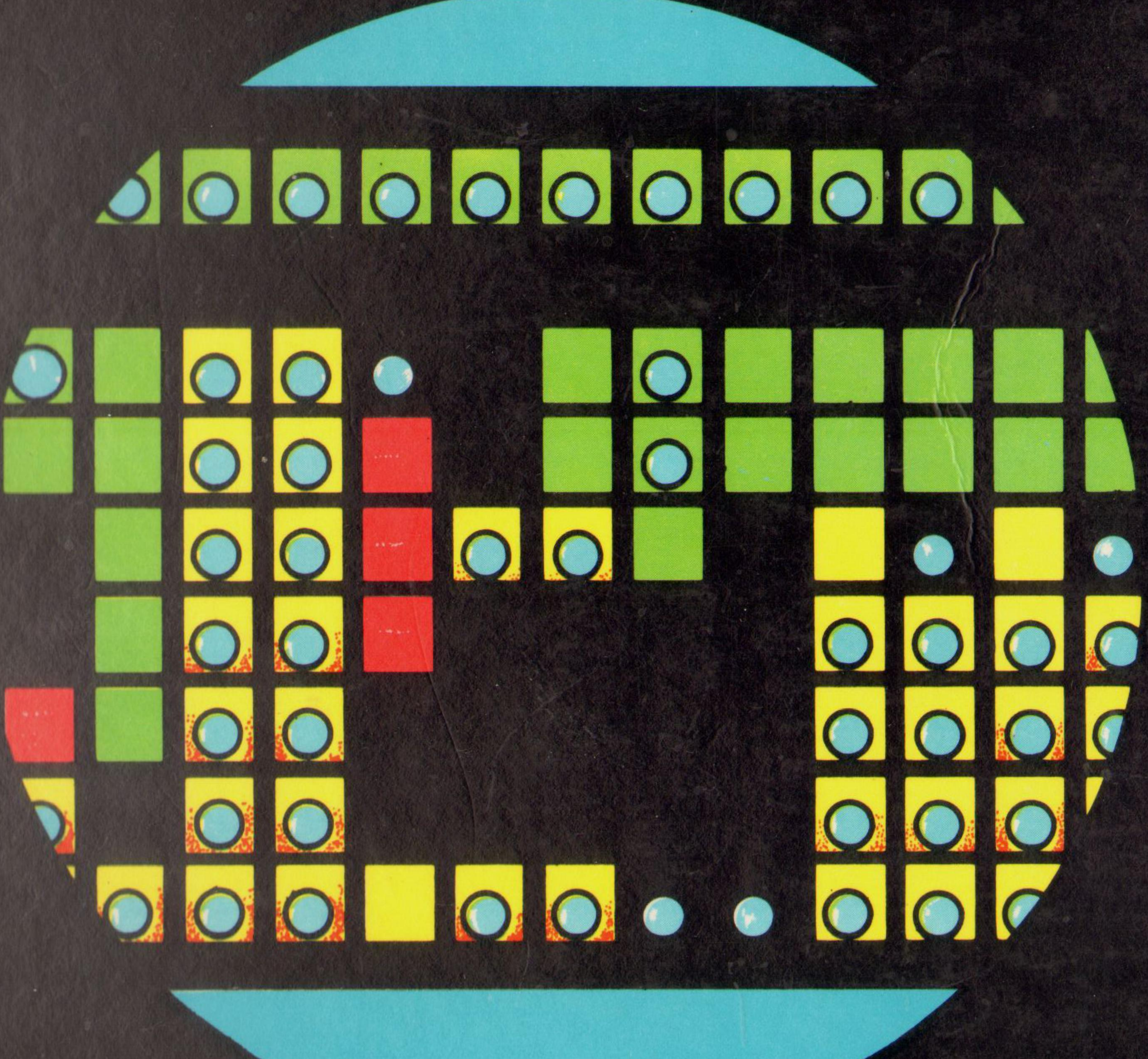
# A számítástechnika legújabb eredményei

## Operációs rendszerek
## Távadatfeldolgozás
## Az időosztásos üzemmód

## 1.



STATISZTIKAI KIADÓ VÁLLALAT

Péter

# A SZÁMÍTÁSTECHNIKA LEGÚJABB EREDMÉNYEI

## 1. KÖTET

# A SZÁMÍTÁSTECHNIKA LEGÚJABB EREDMÉNYEI

SZERKESZTŐ BIZOTTSÁG
DÖMÖLKI BÁLINT
DÖRNYEI JÓZSEF
KÁTAI IMRE
KECSKÉS JÓZSEF
NÁRAY ZSOLT
OBÁDOVICS J. GYULA
VÁMOS TIBOR

A SZERKESZTŐ BIZOTTSÁG
ELNÖKE
PESTI LAJOS

FŐSZERKESZTŐ
QUITTNER PÁL

BAKOS TAMÁS

# Operációs rendszerek

LOHONYAI MIKLÓS

# Távadatfeldolgozás

HUBA ZOLTÁN

# Az időosztásos üzemmód

BUDAPEST, 1974

STATISZTIKAI KIADÓ VÁLLALAT

© Statisztikai Kiadó Vállalat, 1974

# Tartalomjegyzék

Oldal

ELŐSZÓ 9

*Bakos Tamás: OPERÁCIÓS RENDSZEREK*

1. BEVEZETÉS 13
- 1.1. Definició 13

2. AZ OPERÁCIÓS RENDSZEREK SZERKEZETE 18
- 2.1. Feldolgozási és vezérlő programok 19
- 2.1.1. Feldolgozási programok 20
- 2.1.2. Vezérlő programok 21
- 2.1.2.1. System management vagy supervisor 23
- 2.1.2.2. Job management 30
- 2.1.2.3. Data management 37

3. AZ OPERÁCIÓS RENDSZEREK ÁLTALÁNOS PROGRAMOZÁSI ESZKÖZEI 41
- 3.1. Supervisor, egy felhasználóval 41
- 3.2. Több felhasználó rögzitett particiókkal 42
- 3.3. Több felhasználó változó particiókkal 44
- 3.4. A hagymahéj algoritmus 47
- 3.5. Minimális overlay módszer 49
- 3.6. Késleltetett betöltés 53
- 3.7. Lapkezelés (Paging) 55
- 3.7.1. Lapok megosztott használata 58
- 3.7.2. Virtuális tár 62
- 3.7.3. Lapkezeléssel kapcsolatos algoritmusok 63
- 3.7.4. Változó nagyságu lapok 66

4. A GÉPI ARCHITEKTURA ÉS AZ OPERÁCIÓS RENDSZEREK KÖLCSÖNHATÁSA 68
- 4.1. A központi egység hatása 68
- 4.2. A memóriakezelés hatása 69

IRODALOMJEGYZÉK 73

## *Lohonyai Miklós: TÁVADATFELDOLGOZÁS*

BEVEZETÉS ........ 83

1. A TÁVADATFELDOLGOZÁS FEJLŐDÉSI FOLYAMATA, ÁLTALÁNOS PROBLÉMÁI ÉS ALKALMAZÁSAI ........ 84

2. A TÁVADATFELDOLGOZÓ RENDSZER ELEMEI 95

3. AZ ADATVONALAK TULAJDONSÁGAI ........ 99
3.1. Az adatvonalak átviteli sebessége ........ 99
3.2. Vonali üzemmódok ........ 100
3.3. A továbbitott információ felbontása ........ 101
3.4. Analóg és digitális adatátvitel ........ 106
3.5. Az adatvonalak kihasználásának fokozása ........ 109

4. ADATÁTVITELI HÁLÓZATOK JELLEMZŐI ........ 114
4.1. Hálózatok topológiája ........ 114
4.2. Összeköttetések tulajdonságai ........ 116
4.2.1. Közvetlen összeköttetés ........ 118
4.2.2. Felfüzős összeköttetés ........ 119
4.3. Kapcsolt összeköttetések ........ 123
4.3.1. Vonalkapcsolt hálózatok jellemzői ........ 124
4.3.2. Üzenetkapcsolt hálózatok tulajdonságai ........ 125
4.4. Adathálózatok igénybevételi módozatai ........ 128
4.4.1. Nyilvános adathálózatok ........ 129
4.4.2. Független adathálózatok ........ 132

5. KÓDOLÁS ÉS HIBAKORLÁTOZÁS ........ 134
5.1. A modellalkotás néhány kérdése, kódolás ........ 134
5.2. Hibakorlátozás ........ 137

6. AZ ADATÁTVITEL VEZÉRLÉSI ELJÁRÁSAI ........ 147
6.1. Az aszinkron adatátvitel vezérlése ........ 150
6.2. A szinkron adatátvitel vezérlése ........ 151
6.3. Bit-orientált szinkron adatátvitel ........ 158

7. TERMINÁLOK ........ 160
7.1. Terminálok csatlakoztatása ........ 162
7.2. Terminálok jellemzői ........ 164

8. A FELDOLGOZÓ KÖZPONTOK HARDWARE ARCHITEKTURÁJA ........ 168
8.1. Az input-output rendszer ........ 171
8.2. A megszakitási rendszer ........ 174
8.3. Tárkiosztás, tárvédelem ........ 176

9. A KÖZPONTI SZÁMITÓGÉP NÉHÁNY SOFTWARE KÉRDÉSE ---------------------------------- 181
- 9.1. Az ütemező feladatai ---------------------------- 182
- 9.2. Tárkiosztás software oldalról ----------------- 184
- 9.3. A file-kezelő rendszer néhány jellemzője -------- 186
- 9.4. Az adatátviteli programcsomag --------------- 187

10. ADATÁTVITELI KISSZÁMITÓGÉPEK ALKALMAZÁSA ------------------------------------ 192
- 10.1. Programozható adatátviteli vezérlő egységek ----- 192
- 10.2. Intelligens terminálok ------------------------ 196
- 10.3. Adatkoncentrátorok --------------------------- 197

11. A KOMPLEX SZÁMITÁSTECHNIKAI HÁLÓZATOK NÉHÁNY TULAJDONSÁGA -------------------- 200

IRODALOMJEGYZÉK ------------------------------- 205

*Huba Zoltán: AZ IDŐOSZTÁSOS ÜZEMMÓD*

BEVEZETÉS ---------------------------------------- 209

1. BEVEZETÉS AZ ÁLTALÁNOS CÉLU IDŐOSZTÁSOS RENDSZEREKBE ---------------------- 210
- 1.1. Az időosztásos üzemmód kialakulásának okai és az üzemmód definiciója ------------------------ 210
- 1.1.1. Az időosztásos üzemmód kialakulása ----------- 210
- 1.1.2. Az időosztásos üzemmód definiciója ----------- 211
- 1.2. Az időosztásos rendszerek jellemzői és forrásai-- 213
- 1.2.1. Az időosztásos rendszer és az ember ---------- 213
- 1.2.2. Az időosztásos rendszerek formái ------------- 214
- 1.2.2.1. A tulajdonképpeni általános célu időosztásos rendszer ------------------------------------- 215
- 1.2.2.2. Időosztásos tevékenységet lehetővé tevő multiprogramozható kötegelt feldolgozási rendszerek ------ 216
- 1.2.2:3. Speciális célu időosztásos rendszerek ---------- 216
- 1.3. Az időosztásos rendszerek architekturája -------- 216
- 1.3.1. Lényegi és technológiai problémák ------------ 216
- 1.3.2. Az időosztásos rendszerek architekturája ------ 217

2. AZ IDŐOSZTÁSOS RENDSZER VEZÉRLŐ SOFTWAREJÉNEK ALAPVETŐ FUNKCIÓI ----------- 219
- 2.1. A tevékenység és a virtuális processzor fogalma-- 219
- 2.1.1. A tevékenység (process) ---------------------- 219
- 2.1.2. A virtuális processzor ------------------------ 220
- 2.2. Az erőforrások dinamikus allokációja --------- 221
- 2.2.1. Bevezetés ----------------------------------- 221

2.2.2. A memóriagazdálkodás problémaköre 222
2.2.2.1. A logikai és a fizikai cimtartomány összefüggése 222
2.2.2.2. A logikai és a fizikai cimtartomány dinamikus megfeleltetése 223
2.2.2.3. Hierarchikus tárak 227
2.2.2.4. Információcsere (Swapping) 228
2.2.3. A perifériagazdálkodás problémaköre 232
2.2.4. Gazdálkodás a központi egység idejével 233
2.3. A programok helyigényének minimalizálása 234
2.3.1. Reentrant software 234
2.3.2. Szegmentálás 235
2.3.3. Átlapolásos (Overlay) szerkezet 241
2.4. File-rendszer kezelése 241
2.4.1. A file-rendszerrel szemben támasztott követelmények 241
2.4.2. A file-ok tipusai 242

3. AZ IDŐOSZTÁSOS ÜZEMMÓD KÜLÖNLEGES SZOLGÁLTATÁSAI 244
3.1. Interaktiv üzemmód, interaktiv fordítóprogramok 244
3.1.1. Az interaktivitás első foka - mindent a terminálról 244
3.1.2. Az interaktivitás második foka - inkrementális fordítóprogram 247
3.1.3. Az interaktivitás harmadik foka - interpretiv fordítóprogram 250
3.2. Időosztásos üzemmódu előfizetői rendszerek 252

IRODALOMJEGYZÉK 255

TÁRGYMUTATÓ 257

IDEGEN NYELVŰ ÖSSZEFOGLALÓK 263

# Előszó

*A Kormány Számitástechnikai Központi Fejlesztési Programjának végrehajtásához jól képzett számitástechnikai szakemberek ezreire van szükség. E sorozat meginditásával a Központi Statisztikai Hivatalnak, mint a számitástechnika magyarországi alkalmazásáért felelős intézménynek, az a célja, hogy a szakembereknek a számitástechnika egész területét átfogó, magasszintü, a legujabb elvi és gyakorlati eredményeket, illetve alkalmazásokat összefoglaló, önállóan is kerek egészeket alkotó tanulmányokat adjon a kezébe.*

*Az egyes számitógépekben rejlő nagy kapacitások csak akkor realizálhatók kellőképpen, ha a gépeket megfelelő célokra alkalmazzák és megfelelő módon üzemeltetik. Egy kis gépnél talán még megengedhető lehet, hogy néhány percet kihasználatlanul álljon, de egy nagy berendezésnél, melynek percenkénti gépideje több száz forintba kerül, ez már megengedhetetlen pazarlás.*

*Ahhoz, hogy a népgazdasági szinten sok milliárd forint értéket képviselő számitógépeket megfelelő módon használjuk ki, a jó müszaki tulajdonságu gépek mellett jól képzett és a rohamos fejlődés miatt magukat állandóan továbbképezni képes szakemberek kellenek. Ehhez azonban nélkülözhetetlen a megfelelő minőségü, korszerü szakirodalom.*

*Magyar nyelvü irodalom ebben a témakörben viszonylag kevés van, és ezeknek is jelentős része forditás, ami - figyelembe véve a számitástechnika rohamos fejlődését - már megjelenése pillanatában is bizonyos fokig tulhaladott. Ezért választottuk azt a módszert, hogy egy-egy szerző csak viszonylag rövidebb anyagot dolgoz fel, melynek megirása és nyomdai átfutása is viszonylag gyorsan meg-*

*valósitható. Igy az olvasó valóban a számitástechnika legujabb eredményeiről kaphat majd tájékoztatást e sorozat keretében.*

*Minden egyes kötet 3-4 tanulmányt foglal magába, melyek vagy egy-egy nagyobb területről adnak általános áttekintést, vagy egy adott részterületet ismertetnek részletesen az azzal foglalkozó vagy foglalkozni kivánó specialisták számára.*

*A témák és a szerzők kiválasztásában nagy mértékben támaszkodunk a sorozat Szerkesztő Bizottsága tagjainak véleményére, akiknek közremüködésükért ezuton is köszönetet mondok.*

*Bizom abban, hogy minél több mérnök, közgazdász, matematikus és más számitástechnikai szakember találja majd hasznosnak a sorozat folyamatosan megjelenő köteteit.*

*Budapest, 1974. junius*

**Pesti Lajos**

*a Központi Statisztikai Hivatal elnökhelyettese,*
*a sorozat Szerkesztő Bizottságának elnöke*

BAKOS TAMÁS

# Operációs rendszerek

LEKTORÁLTA:
LŐCS GYULA

# 1.

# Bevezetés

A számitógép-tudomány és a számitástechnikai ipar ma is folyó, sok szempontból forradalminak nevezhető fejlődésének egyik terméke a software ipar, melynek kialakulása nagyjából egybeesik a harmadik generációs számitógépek megjelenésével. A software ipar kifejezést a software termékek előállitási technológiájának, méreteinek, forgalmazásának és egyéb olyan tulajdonságainak jellemzésére használjuk, amelyek más - hagyományos - ipari termékeknél is megtalálhatók.

A software ipar termékeinek vizsgálata során első helyen találjuk az operációs rendszereket - mint a többi software termékek (például programozási nyelvek forditóprogramjai, különböző alkalmazásokra orientált programcsomagok stb.) alapját adó software rendszert.

Természetesen az operációs rendszerek software iparban elfoglalt helyének megjelölése még korántsem tekinthető az operációs rendszerek definiciójának, ezt csak egy rövid történeti áttekintés és néhány fogalom tisztázása után adjuk meg.

## *1.1. Definició*

Az első, és részben még a második generációs számitógépek idején, tehát egy-két évtizeddel ezelőtt, az operációs rendszer fogalma nem létezett, de a csirái már az első program-vezérelt gépeknél is megvoltak. Ebben az időben - a hardware (viszonylag) alacsony szintü megbizhatóságának következményeként - az ember és a számitógép kapcsolata is alacsony szintü volt, vagyis a gép üzemeltetői és felhasználói megelégedtek azzal, ha néhány óráig hibátlanul müködő gépet biztositottak. Ebből következően a gépek gyakorlati felhasználását kényelmessé és gazdaságossá tevő eszközök tára - amit ma operációs rendszernek nevezünk - csupán a felhasználást egyáltalán lehetővé tevő eszközökből állt, melyek jórésze hardware uton került megvalósitásra és egy laza, heterogén halmazt alkotott. Az eszközök ezen halmazát - melynek elemei például az üres gépbe való kezdeti programbevitel, a memória tartalmának módositása, a programok futásának adott cim szerinti megállitása stb. - akkor nem nevezték operációs rendszernek és ma is csak a fejlődés érzékeltetésének kedvéért tehetjük ezt.

A mai operációs rendszerek kialakulása szorosan a hardware fejlődését követi. A technikai szinvonal rohamos emelkedése teremtette meg a lehetőséget arra, hogy ne csak egyáltalán "valahogyan", hanem gazdaságosan, kényelmesen használjuk a számitógépeket.

Természetesen a modern operációs rendszerek nemcsak kényelmi és gazdaságossági szempontokat szolgáltak, szükségesek voltak a harmadik generációnál megjelenő fejlett és éppen ezért komplikált hardware biztositotta lehetőségek kihasználása céljából.

A mai operációs rendszerek első definiciójához éppen ezért a számitógépek - pontosabban számitógép-rendszerek különböző, - - eredetileg a felhasználó által létrehozott folyamatok (ld. később) vezérlésére és irányitására szolgáló - funkcióin keresztül juthatunk el /Irod. 1./

A következő funkciókat különböztetjük meg:

- folyamatok létrehozása és megszüntetése;
- a folyamatok végrehajtásának vezérlése, vagyis annak biztositása, hogy minden logikailag erre alkalmas folyamat végrehajtása megtörténjék és egyetlen folyamat végrehajtása sem gátolja korlátlan ideig a többi folyamat végrehajtását;
- a folyamatok végrehajtása során felmerülő különleges helyzetek kezelése;
- a hardware erőforrások szétosztása a folyamatok között;
- software erőforrásokhoz való hozzáférés biztositása a folyamatok számára;
- a számitógép-rendszerben tárolt információ védelmének biztositása és a hozzáférés irányitása;
- a folyamatok közötti információcsere megszervezése és a folyamatok szinkronizálása.

A fenti funkciókat a számitógép-rendszernek kell ellátnia, mivel maguk a folyamatok ezt nem képesek megtenni.

Operációs rendszernek azt a rendszer-softwaret nevezzük, amely a rendszer hardware-jét e funkciók megvalósitásában támogatja.

Az absztrakt definiciók sok előnyük mellett azzal a hátránnyal járnak, hogy a bennük szereplő alapvető fogalmak meghatározása nélkül nem érthetők, avagy könnyen félreérthetőek. Az operációs rendszer definiciójában több, viszonylag egyszerübb fogalom szerepel, ezek ismeretét - a folyamat fogalmának kivételével - feltételezzük.

A folyamat azért képezi további definició tárgyát, mert a szót szokásos jelentésétől bizonyos mértékig eltérő értelemben használjuk: a program fogalmára támaszkodva a folyamatot a program dinamikus tulajdonságokkal rendelkező szinonimájaként definiáljuk, vagyis olyan programként, mely a számitógép-rendszerben létezik és végrehajtásra készen, vagy éppen végrehajtás alatt áll. A program statikus és dinamikus állapotának ilyen módon való szétválasztása több előnnyel jár:

- a végrehajtás alatt álló program, mint folyamat értelmet nyer minden időpillanatban, még akkor is, ha a központi egység éppen nem e program utasitásait hajtja végre;
- kézenfekvővé válik a folyamatok felfüggesztése és folytatása (programoknál azok végrehajtását lehet csak felfüggeszteni vagy folytatni);
- a párhuzamos folyamatok fogalma ugy értelmezhető, hogy adott időpontban több folyamat létezik a rendszerben, amely valamilyen, a kezdeti állapot utáni és a végső állapot előtti állapotban van.

Az operációs rendszer első definiciója mellett, a téma könnyebb megértésének érdekében célszerü azt a felhasználók oldaláról, illetve a kitüzött célok felől is megközeliteni.

A felhasználó számára egy adott számitógép, az operációs rendszerével együtt egy másik, virtuális gépet alkot, amely virtuális gép valamilyen értelemben - a felhasználhatóság szempontjából - jobb, mint az eredeti. A rendszer kialakitása több, egymásra épülő "réteg" formájában történhet /Irod. 2./

Legyen az eredeti hardware gép A0, amelyre a legalsó réteget (egy igen egyszerü operációs rendszert) felépitve, eljutunk egy valamivel jobb A1 géphez. A következő szinten A1-re épitünk egy ujabb réteget (az A1 biztositotta lehetőségek felhasználásával) és igy kapjuk A2-t.

A felhasználó végül valamilyen An gépet használ és általában az eredeti A0-t alig ismeri.

Az operációs rendszerek virtuális gépekkel való felépitése felveti a számitógéprendszer hatékonyságának, teljesitményének kérdését. Hogy mérhető, hogy hasonlitható össze a különböző gépek teljesitménye? Ha a teljesitményt a felhasználó szemével vizsgáljuk, akit nem az érdekel, hogy hány nanosecundum egy gépi összeadás ideje, hanem az, hogy egy feladat megoldásához szükséges programhalmaz mennyi idő alatt hajtható végre, kiderül, hogy az An virtuális gép teljesitménye nem (vagy alig) függ az A0 géphez tartozó utasitások végrehajtási idejétől. Ez más szavakkal azt jelenti, hogy a felhasználó által látott gép teljesitményét döntően az operációs rendszere határozza meg.

Erre jó példa az ICL 1900-as gépcsalád GEORGE 3 nevü operációs rendszere, amelyen egy kb. 140 programból álló csomag segitségével végeztek kisérleteket. /Irod. 3./

Az operációs rendszer egyik változata a várt 90 perces végrehajtási idő helyett a csomagot 82 perc alatt dolgozta fel az 1906A tipusu gépen. A hardware változtatása nélkül, ugyanezen csomagot az operációs rendszer egy fejlettebb változata 55 perc alatt hajtotta végre, azaz csupán az operációs rendszer változtatásával - a gép teljesitményét mintegy 36 %-kal lehetett növelni.

Végül vizsgáljuk meg, mit vár az operációs rendszertől a számitógép üzemeltetője, a számitóközpont vezetője, akinek az a célja, hogy a nagy tőkebefektetést igénylő rendszer üzletileg rentábilis legyen, gazdaságosan üzemeljen. Az eddig elmondottak alapján is várható, hogy az operációs rendszernek nagy szerepe van a számitógép-rendszer üzemeltetésének gazdasági vonatkozásu kérdéseiben és ezt csak alátámasztják a rendszer hardware és software komponenseinek értékarányára vonatkozó legujabb becslések. Ezek szerint /Irod. 4./ a 60-as évek elején a számitógép-rendszerek értékében a hardware-software értékarány 70%-30% volt, ugyanez jelenleg 50%-50%-ra tehető és a jövőben további - a software komponensek javára való - értékeltolódás várható.

A számitógép üzemeltetője az operációs rendszertől olyan szolgáltatásokat vár, amelyek lehetővé teszik a gép gazdaságos felhasználását. Ez, elsősorban a bérmunkában dolgozó számitóközpontok esetében, a felhasználók és a számitógép (pontosabban az operációs rendszer által létrehozott virtuális gép) kapcsolatának olyan szintre emelését jelenti, ami biztositja a számitóközpont kapacitásának jó kihasználását. Ezen tulmenően azonban többféle mutató alapján vizsgálhatjuk, illetve optimalizálhatjuk a számitógép kihasználását.

A legkézenfekvőbb stratégia, hogy a rendszer müködése közben a hozzátartozó hardware eszközök közül minél kevesebb legyen kihasználatlan, azaz lehetőleg mindegyik állandóan dolgozzék. Ez a stratégia az alapja az ugynevezett *kötegelt (batch)* üzemmódban dolgozó rendszereknek, ahol az elvégzendő munka előre rögzitett "feladat kötegek" formájában érkezik a számitóközpontba, az elvégzett munka eredményei viszonylag ritkán jutnak vissza a felhasználóhoz és a munka tényleges elvégzésének idején a gép és a felhasználó között általában nincs "visszacsatolás". Mivel a kötegelt üzemmódnál egy-egy feladat végrehajtására nem egy konkrét időpont, hanem nepos nagyságrendü időintervallum áll rendelkezésre, az operációs rendszernek elvileg lehetősége van a feladatok olyan csoportositására, ami biztositja a számitógép előbb emlitett optimális kihasználását.

Az operációs rendszerek másik fajtája - ezzel a továbbiakban nem foglalkozunk - az *időosztásos* rendszer, amelyben a felhasználókat párhuzamosan, interaktiv módon kell kiszolgálni és a feladat megoldásához a felhasználó a számitógéppel folytatott "párbeszéd" során jut el. Az időosztásos (time-sharing) elnevezés arra utal, hogy az operációs rendszer egy olyan virtuális gépet ad a felhasználók kezébe, amelyet azok (látszólag) egyszerre sokan, az általuk választott időpontban tudják interaktiv módon használni. Tehát a sulyponti probléma a központi egység idejének szétosztása a felhasználók között.

A kötegelt és az időosztásos rendszerek természetesen még sok tulajdonságukban különböznek egymástól, az alapvető eltérés azonban ugy foglalható össze, hogy a kötegelt rendszer elsődleges célja a számitógép különböző egységeinek állandó munkában tartása, az időosztásos rendszer viszont elsősorban a felhasználó közvetlen géphez fordulását biztositja, hogy minden felhasználó ugy érezze, egyedül használja a rendszert.

Mi a továbbiakban a kötegelt üzemmódban dolgozó operációs rendszerekkel foglalkozunk, megjegyezvén, hogy ez sok esetben nem választható élesen el az időosztásos üzemmódtól, például annak következtében, hogy az interaktiv géphasználat intenzitása időben erősen változó (függ a napszakoktól) és ezért a legtöbb számitóközpontban a kötegelt és az időosztásos üzemmódot vegyesen (általában párhuzamosan) alkalmazzák. (Meg kell azonban jegyeznünk, hogy a kötegelt üzemmódu rendszerek csak korlátozott számu interaktiv felhasználó kiszolgálására képesek.)

A 2. fejezet az operációs rendszerek szerkezeti felépitésével foglalkozik az átlagos felhasználó számára szükséges szintnél valamivel részletesebben, különösen a job control (munkavezérlő) nyelv vonatkozásában.

A 3. fejezet az operációs rendszerekben a tárral való gazdálkodásra használt általános, jórészt programozási eszközök áttekintését adja és a 4. fejezettel együtt, amely az operációs rendszerek és a gépi architektura kölcsönhatását igyekszik megvilágitani, elsősorban rendszer fejlesztőknek (tervezőknek és programozóknak) szól. Segitséget nyujthat azonban minden felhasználónak az operációs rendszerek "lelkivilágáról" alkotott nézetei kialakitásához.

## 2.

# Az operációs rendszerek szerkezete

Az operációs rendszerek szerkezeti felépitésének tárgyalásához szükségünk lesz néhány, egymáshoz szorosan kapcsolódó fogalom meghatározására. Mindenekelőtt a program és a folyamat által alkotott fogalompárhoz kapcsoljuk a *job* fogalmát, ami egy vagy több, általában logikailag összetartozó programokból álló, a számitógépen való végrehajtásra előkészitett munkaegység.

Egy job-ot alkothat például három FORTRAN program forditása, egyetlen futtatható programmá való szerkesztése - ami tartalmazza a szükséges könyvtári rutinok beépitését is -, ennek felvétele a felhasználói programkönyvtárba, végül innen való betöltése és futtatása.

A fenti példából is látható, hogy egy job általában több lépésből - *job-step*-ből - tevődik össze, minden job-step egy-egy önállóan végrehajtandó programot jelent a job-on belül.

Mig a job - amint ezt később látni fogjuk - egy speciális nyelven irott program formájában megjelenő, jól körülhatárolt egység, a job-step-eket egy adott job-on belül többé-kevésbé önkényesen választhatjuk meg, aszerint, hogy milyen mélységü elemzésre van szükség. Például egy FORTRAN program forditása tekinthető egyetlen job-stepnek, de felbontható a forditóprogram betöltésére és futtatására, sőt mindkét lépés még további kisebb tevékenységekből is összerakható, amelyek még mindig önálló programok végrehajtását jelentik.

A job-ok segitségével kifejezve, az operációs rendszer feladata, hogy a felhasználók által rendelkezésére bocsátott job-halmazt valamilyen - a rendszer gazdaságos kihasználását biztositó - stratégia szerint végrehajtsa.

Egy job végrehajtásának szükséges - de nem elegendő - feltétele, hogy az igényelt eszközök rendelkezésre álljanak. A végrehajtás eszközeit közös néven ***erőforrásoknak*** nevezzük és két nagy csoportba soroljuk:

- hardware erőforrások,
- software erőforrások.

A hardware erőforrásokhoz tartozik a központi egység, az operativ memória, a háttértárak, az input-output és egyéb perifériális készülékek stb., vagyis minden - a számitógép-rendszer munkájában résztvevő - önálló hardware eszköz.

A software erőforrások az operációs rendszer keretében müködő programok, amelyek résztvesznek a felhasználó job-jának végrehajtásában. Az előző példához kapcsolódva, software erőforrás például a FORTRAN forditóprogram vagy a linkage-editor (szerkesztőösszekapcsoló program).

Minden job-hoz egy változó, de minden időpillanatban jól meghatározott *erőforrás-igény* tartozik, és az operációs rendszer egyik alapvető feladata, az erőforrásokkal való gazdálkodás. A job végrehajtása csak akkor kezdődhet meg - illetve folytatódhat - ha pillanatnyi erőforrás-igénye kielégithető, másszóval ha az erőforrás-igény menetközben ugy változik, hogy az egy adott időpontban már nem elégithető ki, a job végrehajtása félbeszakad. Bizonyos értelemben az idő is valamilyen különleges erőforrásnak tekinthető, ha például előirjuk, hogy egy job legfeljebb mennyi ideig használhatja a központi egységet, mert ilyenkor a job végrehajtásának (folytatásának) az is feltétele, hogy még nem használta fel a számára megengedett időt.

Ezek után az operációs rendszerek szerkezetét kivülről - a nagyobb egységek irányából - befelé haladva közelitjük meg.

## 2.1. *Feldolgozási és vezérlő programok*

Az operációs rendszer komponenseit első közelitésben két nagy csoportba sorolhatjuk. Az egyik csoportot *feldolgozási* programoknak, a másikat *vezérlő* programoknak nevezik. A bevezetésben kifejtett elveknek megfelelően, a vezérlő programokkal fogunk részletesen foglalkozni, ami azonban nem a feldolgozási programok szerepének lebecsülése, hanem annak a ténynek a következménye, hogy a felhasználók a feldolgozási programokkal sokkal közelebbi kapcsolatban állnak, azokat - vagy legalábbis a gyakran használt részüket - jobban ismerik, mint a "be nem avatottak" számára néha misztikusnak tünő vezérlő programokat.

Egy-egy felhasználó általában egy, vagy néhány programozási nyelvet használ és ezen nyelvnek forditóprogramjait a saját szemszögéből a gyakori használat következtében nagyon jól megismeri. Tudja, hogy milyen utasitás sorozatból keletkezik jó hatásfoku tárgykód, tudhat a forditóprogram hibáiról, esetleg olyan tulajdonságairól, melyek a hivatalos specifikációban nem szerepelnek, de előnyösen kihasználhatók (pl. a kulcsszavakat nem kell végig kiirni, mert a forditóprogram az első néhány betü alapján ismeri fel azokat) és igy tovább.

Ugyanakkor a felhasználónak a multiprogramozás szervezéséről vagy az erőforrásokat kiosztó rutinokról - a velük való közvetlen kapcsolat hiánya, vagy egyoldalusága következtében - általában sokkal ködösebb elképzelései vannak.

### 2.1.1. *Feldolgozási programok*

A feldolgozási programok a programozási nyelvek forditóprogramjaiból, alkalmazási programok bizonyos részeiből, valamint segédprogramokból (utility) állnak.

A forditóprogram elnevezést csak az egyszerüség kedvéért használtuk, tulajdonképpen a különböző programozási nyelvek implementációjáról, azaz gépi feldolgozásukat megvalósitó programokról van szó, és ilymódon ide tartoznak az értelmezési, vagy forditási és értelmezési technikát vegyesen alkalmazó programok is. Létezik a forditóprogramok más szempontból való felosztása is, aszerint, hogy milyen szintü nyelv feldolgozását végzik. Például a gépi kódhoz közelálló, un. assembly nyelvek forditóprogramját assemblernek hivják, a speciális célu, magasszintü nyelvek feldolgozásánál bizonyos esetekben előtét forditókat (preprocessor) alkalmaznak.

Az operációs rendszer, illetve az általa kialakitott virtuális gép szinvonalának egyik mutatója a hozzátartozó forditóprogramok minősége és száma.

Az alkalmazási programok olyan heterogén halmazt alkotnak, hogy téma szerinti részletesebb csoportositásukat meg sem kisérelhetjük.

Az alkalmazott technikai megoldások szempontjából általában *paraméteres programokat* és *program generátorokat* különböztetünk meg.

A paraméteres program - általában program rendszer - valamely feladat általános megoldásához szükséges tipusprogram modulok halmaza, amelyből egy konkrét esetben a felhasználandó modulok kiválasztásához és müködéséhez az információt paraméterek formájában kell megadni. (Például lineáris programozási, matematikai statisztikai program-rendszer.)

A program generátorok (pl. rendezőprogram generátor) szintén paramétereket használnak, ezek alapján azonban a feladat megoldását nem közvetlenül végzik el, hanem első lépésben eredményül egy olyan programot adnak, amelyik képes a feladat megoldására. A legtöbb esetben a generált program még egy forditóprogram által kerül leforditásra, mielőtt futtatható lenne. A generált program előnye, hogy az adott feladatra szabott, nem tartalmaz a konkrét számitásnál felesleges, nem igényelt modulokat. Ezen kivül, a generált program megőrizhető és azonos, vagy hasonló alkalmazásnál az ujra generálásra nincs szükség.

Hátránya, hogy a feladat tényleges megoldását elvégző programhoz több előkészitő lépésre van szükség.

Az alkalmazási programok készletének bővitésére a felhasználóknak is lehetőségük van, ami azért nagy jelentőségü, mert a felhasználók tudják legjobban felmérni a rendszerrel szemben támasztott igényeiket.

A rugalmas, jól szervezett operációs rendszerekben éppen ezért egyszerü, de hatékony eszközöket biztositanak arra, hogy a felhasználók által irott programokat a rendszerbe beépithessék. A beépités gyakorlatilag a program felvételét jelenti a rendszer megfelelő könyvtárába. A könyvtárak kezelését már meglévő vezérlő és feldolgozási programok végzik, tehát a rendszer ilymódon saját magának bővitésére képes.

Csak nagyon röviden foglalkozunk a feldolgozási programok harmadik csoportjával, a segédprogramokkal. Szerepük az operációs rendszeren belül a rendszer gyakorlati alkalmazásának megkönnyitése. Ide tartoznak a file-okat egyik információhordozóról a másikra átmásoló programok, a szövegszerkesztők, a könyvtárak katalógusát listázó és sok más olyan program, melyekkel csak a gép kezelői kerülnek közvetlen kapcsolatba.

### *2.1.2. Vezérlő programok*

Az operációs rendszerek magját alkotó vezérlő programok legfontosabb funkciói a következők:

- a rendszer folyamatos müködésének biztositása,
- optimális gazdálkodás az erőforrásokkal,
- a job-ok prioritásainak kezelése.

A fenti funkciók részletes tárgyalására a vezérlő programok szerkezetének vizsgálatakor térünk ki, most csak a *prioritás* fogalmát tisztázzuk.

A job prioritása kettős célt szolgáló mérték, amit általában egy természetes szám vagy betü reprezentál.

Az egyik cél a job-hoz tartozó folyamatok fontosságának kifejezése, amit a felhasználó, illetve a gép üzemeltetője határozhat meg. A másik - ezzel szorosan összefüggő - cél a multiprogramozás szervező algoritmusának támogatása. Abban nyilvánul meg, hogyha egyéb feltételek teljesülése következtében több folyamat indulhatna, ezek közül a legmagasabb prioritásu kerül elinditásra és csak akkor kell más (általában szekvenciális) alapon választani, ha több ilyen van.

A prioritás kezelésére sokféle módszert alkalmaznak, melyeknek közös lényege, hogy a kezdeti prioritást a job-ra vonatkozó egyéb információval együtt a felhasználó adja, a job végrehajtása során pedig a prioritása automatikusan módosulhat, pl. attól függően, hogy hányszor "maradt alul" a job, az erőforrásokért folyó verseny folyamán.

A vezérlő programok első közelitésben három nagy csoportra oszthatók:

a) *system management*, melynek rutinjai logikai kapcsolatot létesitenek a hardware és az operációs rendszer többi része között;

b) *job management*, melynek rutinjai logikai kapcsolatot létesitenek a vezérlő programok és a feldolgozási programok között;

c) *data management*, melynek rutinjai software kapcsolatot létesitenek a feldolgozási programok és a számitógép-rendszerhez tartozó külső tárolók között.

Mindhárom csoport további részekre oszlik, ezek megértéséhez azonban a gépi architekturához kapcsolódó néhány fogalommal kell megismerkednünk.

A számitógép-rendszer központi egysége két *állapotban* működhet: normál vagy speciális állapotban. A két állapot általában hardware által realizált, de sok esetben software segitséggel kerül megvalósitásra.

A speciális állapotot a system management rutinok számára tartják fent, és mivel ezeket közös néven *supervisor*[1]-nak nevezik, a speciális állapot neve *supervisor állapot*.

A supervisor állapotban a folyamatok hozzáférhetnek a teljes operativ memóriához és tartalmazhatnak olyan un. privilegizált utasitásokat (pl. i/o utasitások), melyek a normál állapotban működő programokban nem szerepelhetnek.

A normál állapotban futó programok további két csoportra oszthatók: privilegizált és nem privilegizált programok. A privilegizált programok hozzáférhetnek a memória bizonyos speciális részeihez, például a rendszer katalógushoz (ld. később, a data management tárgyalásánál). Nem privilegizált programból való hivásuk csak a supervisor-on keresztül történhet. A fejlettebb operációs rendszerekben a privilegizált rutinokat a job-ok közösen használják.

A feldolgozási programok (például programozási nyelvek forditó programjai) általában nem privilegizált programok.

A másik alapvető, a hardware és a software határterületén létező fogalom a *megszakitás*, ami tulajdonképpen a supervisor vezérlésére szolgál. A megszakitások különböző események bekövetkezésének eredményei, általában hardware indikátorok beállitásával kerülnek realizálásra. Közös tulajdonságuk, hogy a rendszert normál állapotból supervisor állapotba viszik át (ha ti. a rendszer nem volt előzőleg egyébként is supervisor állapotban).

A megszakitások vezérlő szerepe abban nyilvánul meg, hogy egy adott folyamat addig folytatódik, ameddig vagy

- valamilyen supervisor szolgáltatást nem igényel, pl. fizikai szintü I/O, vagy
- valamilyen esemény bekövetkezése miatt egy supervisor szolgáltatásra van szükség (pl. beavatkozott a gép kezelője).

Mindkét esetben a folyamat megszakad, a gép supervisor állapotba kerül, a vezérlés a megszakitást elemző supervisor rutinnak adódik át. A megszakitás kezelés olyan módon zajlik le, hogy a felfüggesztett folyamat később folytatható legyen (ez a folyamat pillanatnyi állapotának tárolását jelenti).

[1] Közelitő jelentése: felügyelő program. Más, idegen nyelvü szinonimái: monitor, executive stb.

A megszakitások a következő tipusokba sorolhatók:

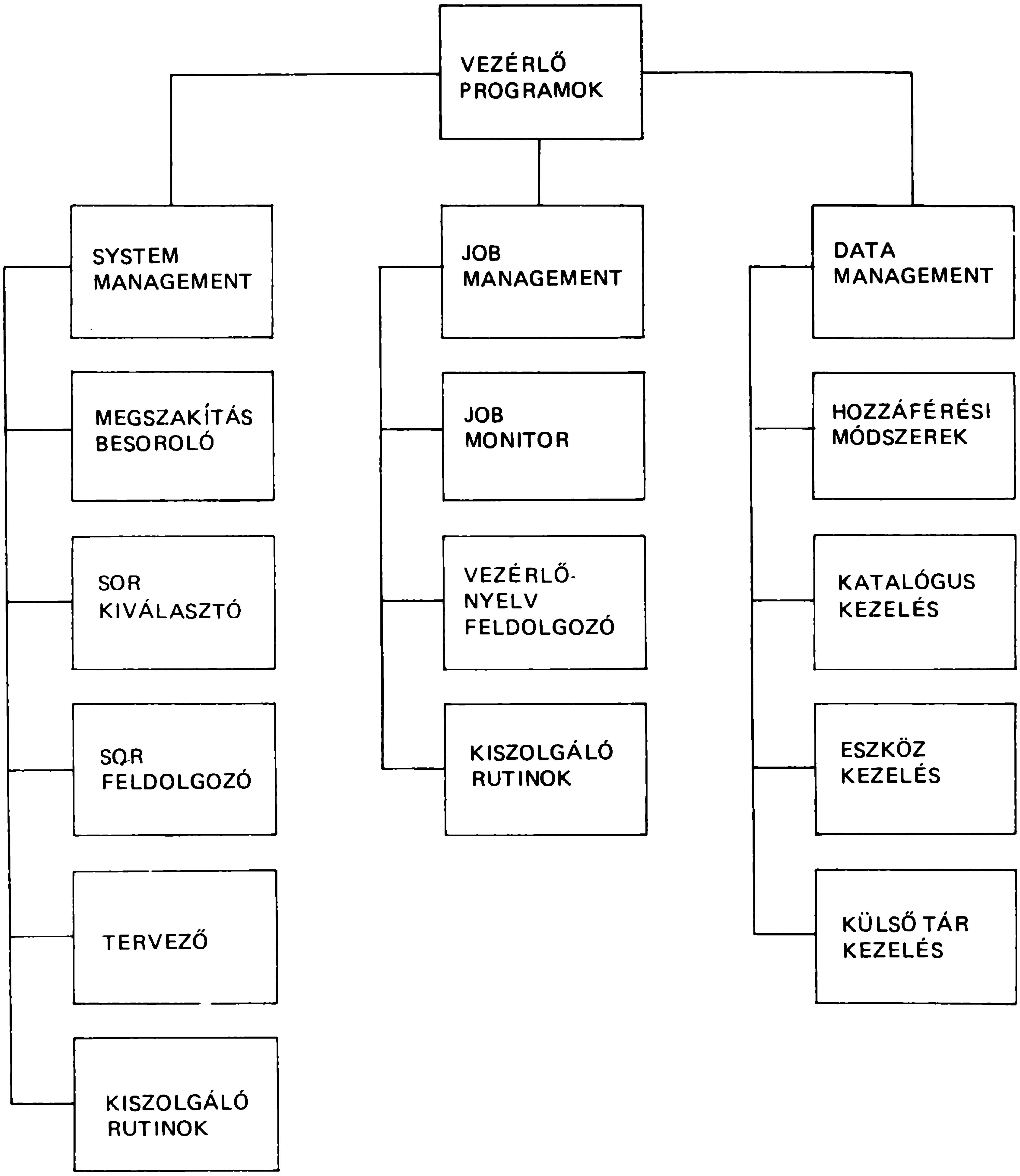
- Supervisor hivás, amit egy általában normál állapotban működő programban végrehajtott, erre a célra szolgáló utasitás eredményez. A supervisor hivás általában akkor használatos, ha a futó program valamilyen supervisor szolgáltatást igényel.
- Program megszakitás, ami automatikusan generálódik egy gépi utasitás végrehajtásakor előálló, szokatlan esemény (programhiba) hatására. Ilyen megszakitás jöhet létre pl. akkor, ha az utasitás a programhoz rendelt cimtartományon kivül eső cimre hivatkozik.
- Géphiba megszakitás, amit a hardware működésében észlelt hiba okoz (pl. memória paritás hiba).
- Input-output megszakitás, amit az I/O egységek generálnak annak jelzésére, hogy a pillanatnyilag folyamatban lévő adatátvitellel kapcsolatban valamilyen feltétel teljesült, vagy esemény bekövetkezett. (Például az I/O egység nem működőképes, vagy az átvitel befejeződött.)
- Külső megszakitás, amit a géphez kapcsolt valamilyen külső egység okoz. Az egyik tipikus külső egység a konzol irógép, ennek segitségével kezdeményezhetők a gépkezelő beavatkozásai, a különböző speciális alkalmazásoknál (pl. folyamat-vezérlés), azonban igen sokféle külső megszakitást kiváltó készülék használatos. Ezek egyik leggyakoribbja az óra (timer).

A vezérlő programok szerkezetét az 1. ábrán mutatjuk be, az egyes rutinok (Megszakitás besoroló, Sorkiválasztó stb.) ismertetése később következik.

### *2.1.2.1. System management vagy supervisor*

Az ide tartozó rutinok alapfunkciója a megszakitások feldolgozása. Ezzel kapcsolatban az első feladat a megszakitások tipus szerinti osztályozása, majd pedig a megfelelő megszakitás feldolgozó rutin meghivása. Mivel ezek a feladatok a rendszer hatékony működéséhez nélkülözhetetlenek, a supervisor rutinjai több szempontból is megkülönböztetett helyzetben vannak, nemcsak a felhasználói, de még a többi rendszer programhoz képest is. Ennek megfelelően a supervisor rutinok:

- általában rezidensek az operativ memóriában, hogy gyorsan és kényelmesen aktivizálhatók legyenek:
- feldolgozási programokból közvetlenül nem hivhatók, mivel működésüket a megszakitások vezérlik;
- supervisor állapotban működnek, a teljes hardware hatékony kezelése érdekében;
- a rendszerben csak egy aktiv példányban léteznek.


VEZÉRLŐ PROGRAMOK
SYSTEM MANAGEMENT
JOB MANAGEMENT
DATA MANAGEMENT
MEGSZAKÍTÁS BESOROLÓ
JOB MONITOR
HOZZÁFÉRÉSI MÓDSZEREK
SOR KIVÁLASZTÓ
VEZÉRLŐ-NYELV FELDOLGOZÓ
KATALÓGUS KEZELÉS
SOR FELDOLGOZÓ
KISZOLGÁLÓ RUTINOK
ESZKÖZ KEZELÉS
TERVEZŐ
KÜLSŐ TÁR KEZELÉS
KISZOLGÁLÓ RUTINOK

*1. ábra*
*A vezérlő programok diagramja*

A supervisor másik alapvető tevékenysége a job-ok végrehajtásra való előkészitése, ami azt jelenti, hogy a felhasználó által a job-hoz adott információ alapján, jól kezelhető nyilvántartást kell vezetni. Ebben a nyilvántartásban minden job-ot egy egyértelműen strukturált információ-halmaz, a *job vezérlő blokk* képvisel.

A job vezérlő blokkban a következő fontosabb adatok szerepelnek:

- a job azonositója;
- prioritás;
- a felhasználó által adott job control információt hordozó input eszköz azonositása;
- a job pillanatnyi állapotát mutató jelzők (pl. végrehajtás közben, végrehajtásra készen, I/O művelet miatt felfüggesztve stb);
- tipus jelző, arra a job-ra vonatkozó megszakitás számára, amelynek feldolgozása éppen folyik;
- a job egy-egy megszakitás utáni folytatásához szükséges hardware regiszterek tartalmának kimentésére szolgáló munkaterület;
- a job-ra vonatkozó, még kiszolgálásra váró megszakitások jelzői;
- a job által felhasznált gépidő számlázását lehetővé tevő adatok.

A job vezérlő blokkot kizárólag a supervisor rutinok kezelik, a felhasználó job-jából nem hozzáférhető. Minden job-hoz a rendszerben egy job vezérlő blokk tartozik. Mivel a job vezérlő blokkok a rendszer működése folyamán állandóan létrejönnek és megszűnnek, tárolásukat sokszor mutatók segitségével láncolt lista-szerkezettel valósitják meg. Ilyenkor természetesen minden vezérlő blokk két mutatót is tartalmaz, amelyek az előző és a következő blokkra mutatnak.

A supervisor alapvető funkcióit néhány fontos rutin valósitja meg (ld. 1.ábra). Ezek:

- megszakitás besoroló,
- sorkiválasztó,
- sor feldolgozó,
- tervező és
- kiszolgáló rutinok.

A megszakitás besoroló feladata a megszakitások osztályozása és a különböző tipusu - még ki nem szolgált - megszakitásoknak sorokban való elhelyezése.

A megszakitások sorbaállitására azért van szükség, mert a párhuzamosan létező folyamatokból érkező megszakitások gyakorisága olyan nagy lehet, hogy többszörösen meghaladja a supervisor rutinok megszakitás feldolgozó képességét. Ne felejtsük el, hogy a supervisor rutinok csak egy példányban léteznek, a megszakitások pedig több párhuzamos folyamatból érkezhetnek. Továbbá, hogy feldolgozásuk általában több rutin összehangolt munkájának eredménye.

A különböző tipusu megszakitások soraiban az elemek - valamilyen jól definiált formában - minden olyan információt tartalmaznak, ami az adott tipusu megszakitás további feldolgozásához szükséges. Ebből következően a sor elemek a különböző sorokban különbözőek, azonban mindegyik tartalmaz egy, a job vezérlő blokkra való mutatót, valamint a sor elemek egymáshoz láncolásához szükséges mutatókat.

Miután egy megszakitás a megfelelő sorban elhelyezésre került, a megszakitás besoroló a következő tevékenységeket végzi:

- Ha a megszakitás normál állapotban történt, az aktiv job folytatási paraméterei kimentésre kerülnek a job vezérlő blokkba és a vezérlés a sorkiválasztó rutinnak adódik át.
- Ha a megszakitás supervisor állapotban következett be, a vezérlés a megszakitás helyére adódik vissza, hogy a supervisor rutinok a munkájukat helyes sorrendben folytathassák. Ebben az esetben a megszakitás besoroló csak "kimentette" a megszakitást a későbbi feldolgozás számára.

A megszakitás feldolgozás állapot diagramját a 2. ábra szemlélteti.

Az ábrán a hiba állapotot csak a jobb áttekinthetőség kedvéért vezettük be. Ha a rendszer az E állapotba kerül, csak külső beavatkozással mozditható ki onnan.

A sorkiválasztó rutin alapvető feladata a megfelelő sorfeldolgozó rutin kiválasztása és elinditása. A rutin egy sorrend táblázaton operál, melynek elemei a megszakitások prioritási sorrendjében helyezkednek el. A sorrend táblázatban minden megszakitás sorhoz általában egy elem tartozik. Az I/O megszakitások soraihoz a különböző perifériáknak megfelelően több elem is tartozhat.

A sorrend táblázatot és egy megszakitási sort mutat be a 3. ábra.

A sorkiválasztó sorban megvizsgálja a sorrend tábla elemeit. Ha ugy találja, hogy valamelyik elemhez nem üres megszakitási sor tartozik, a megfelelő sorfeldolgozó kerül hivásra, ha pedig nincs elvégzendő supervisor tevékenység, a vezérlés a tervezőnek adódik át, amely egy feldolgozási programot választ ki és indit el.

A tervező (ütemező) feladata a végrehajtandó job kiválasztása, a megszakitások továbbitása a job management rutinoknak, valamint a kiválasztott job inditása.

A végrehajtandó job kiválasztása prioritásos, szekvenciális, vagy időosztásos [1] alapon történhet, az erre használatos algoritmusok rendszerenként mások és mások.

Végrehajtásra való kiválasztás szempontjából csak az a job jöhet számitásba, melynek job vezérlő blokkjában az állapot jelző "végrehajtás közben" vagy "végrehajtásra készen" állapotot mutat.

---

[1] Az időosztás egyik leggyakoribb realizálása az un. "time-slicing", ami állandó "időszeleteket" biztosit a job-ok számára.

A kiválasztott job inditása gyakorlatilag azt jelenti, hogy a vezérlést az utolsó megszakitás helyét követő utasitásnak kell átadni. Ettől eltérő módon kell kezelni azokat a program megszakitásokat, amikor is a felhasználó programja nem folytatható, helyette a vezérlést a megszakitásra vonatkozó egyéb adatokkal együtt a megfelelő job management rutin kapja. Ez azután gondoskodik a program megszakitás további feldolgozásáról, ami tartalmazza például a gépkezelőnek a programhiba körülményeiről való értesitését is.

A job-ok befejezése ("job terminálás") külön emlitést érdemlő téma, ahol két alapesetet különböztethetünk meg:

- normális befejezés (lásd 2.2.2.1. fejezet),
- nem normális befejezés.

Mindkét eset kezelésére külön szubrutin csomag szolgál, az elvégzendő lényegesebb teendők a következők:

- a gépkezelő értesitése a job befejezésének okáról és körülményeiről,
- a gépfelhasználás számlázásához szükséges tevékenységek,
- a le nem zárt file-ok lezárása,
- a job vezérlő blokk adatainak módositása.

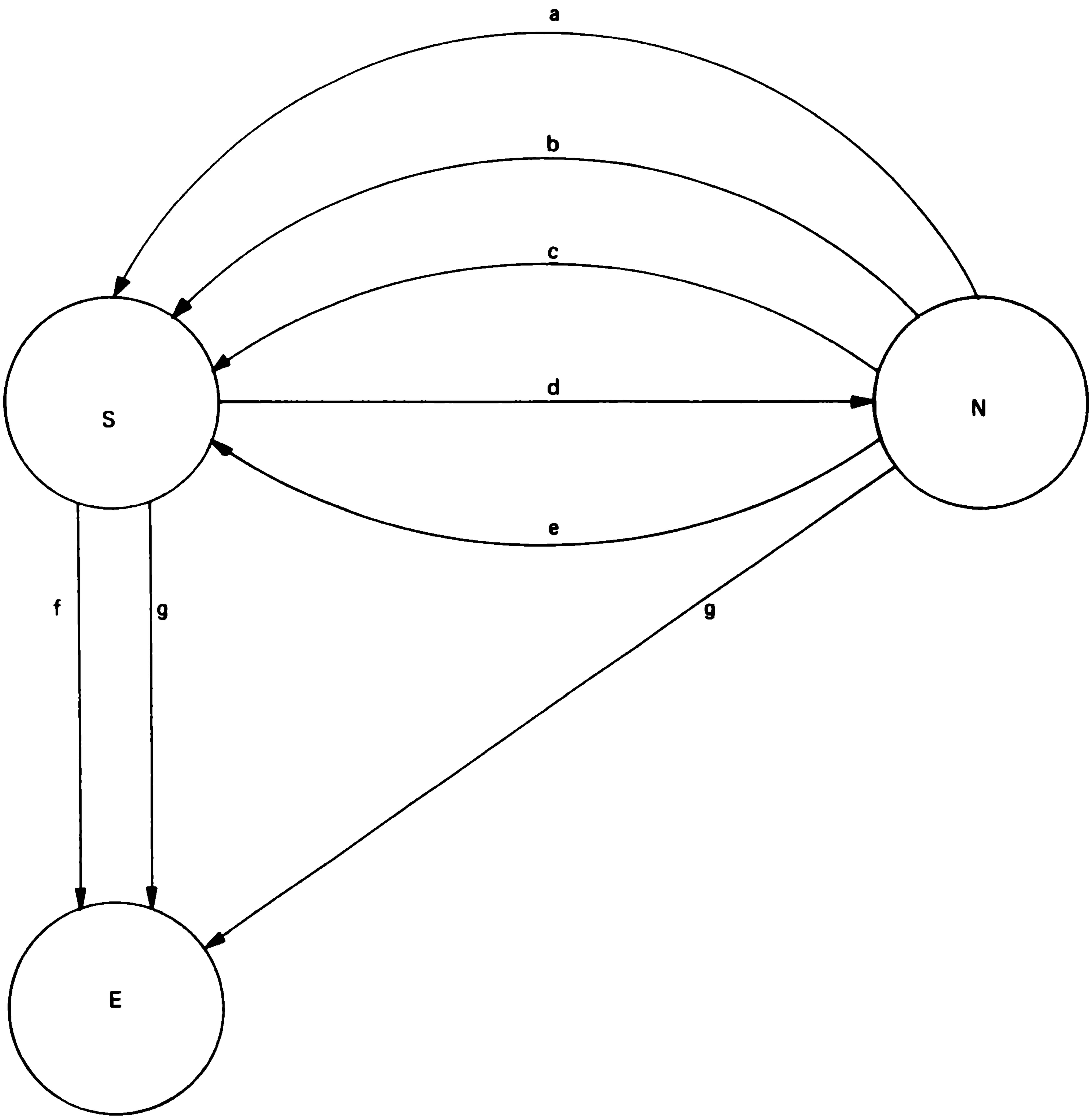
A sorfeldolgozó feladata az általa kezelt megszakitás sor elemeinek feldolgozása. Általában minden sorelem feldolgozása után a sorkiválasztó kerül ujra hivásra, hogy a megszakitások prioritás szerinti kiszolgálása biztositva legyen. Előfordulhat azonban, hogy egy megszakitás feldolgozása több sor-feldolgozó rutin működését igényli.

Legyen például az S1 megszakitási sorban egy E1 elem olyan, hogy feldolgozásához szükség van az S2 megszakitási sor feldolgozójának működésére is. Ekkor E1 feldolgozása azt eredményezi, hogy S2-ben a megfelelő időben való feldolgozás céljából egy uj elem jelenik meg.

Egy-egy sorelem feldolgozása folyamán a sorfeldolgozó módositja a job vezérlő blokk tartalmát, input-output müveleteket kezdeményez és kiszolgáló rutinokat hiv.

A kiszolgáló rutinok szerepe különösen a supervisor hivások esetében szembetünő, ahol ezek alapvető supervisor szolgáltatásokat valósitanak meg:

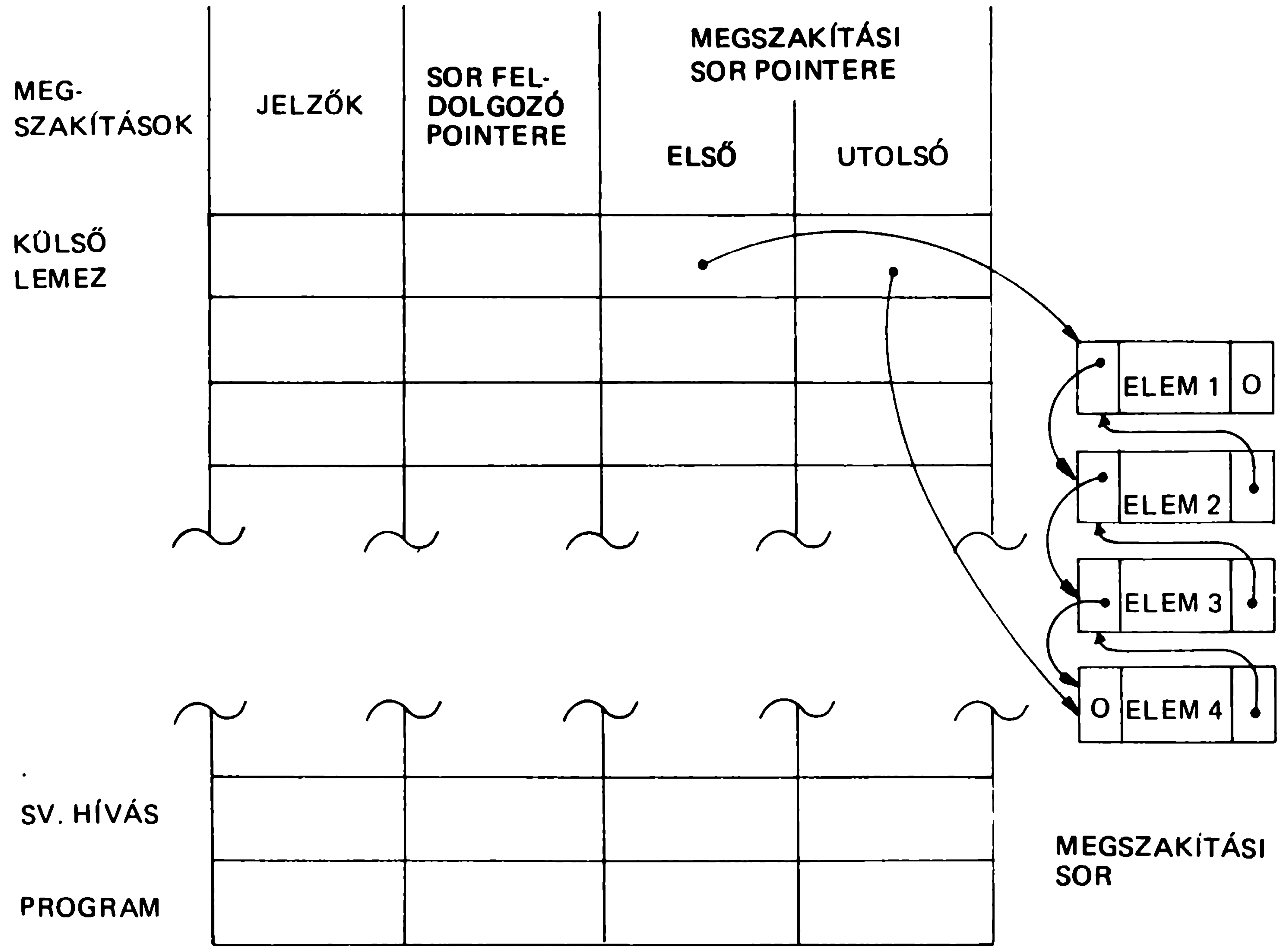
- Memória foglalás. Az operativ tár igényelt nagyságu részének a job-hoz rendelése.
- Input-output kezdeményezés.
- Vezérlés átadás, privilegizált programokból nem privilegizált programoknak, és forditva.
- A rendszer órájának egy job számára való beállitása és leolvasása.
- Egy job-hoz tartozó job vezérlő blokkból való információ kiolvasás (csak privilegizált programok számára hozzáférhető).
- A job prioritásának beállitása és változtatása (csak privilegizált programok számára hozzáférhető).



Jelölések:

S: supervisor állapot
N: normál állapot
E: hiba állapot
a: program megszakítás normál állapotban
b: supervisor hívás
c: I/O megszakítás
d: felhasználói program indítása/folytatása
e: külső megszakítás
f: program megszakítás supervisor állapotban
g: géphiba megszakítás

*2. ábra*
*Megszakitás feldolgozás állapot diagramja*


MEG-
SZAKÍTÁSOK
JELZŐK
SOR FEL-
DOLGOZÓ
POINTERE
MEGSZAKÍTÁSI
SOR POINTERE
ELSŐ
UTOLSÓ
KÜLSŐ
LEMEZ
ELEM 1
O
ELEM 2
ELEM 3
O
ELEM 4
SV. HÍVÁS
PROGRAM
MEGSZAKÍTÁSI
SOR

*3. ábra*
*Sorrend táblázat és megszakitási sor*

### *2.1.2.2. Job management*

A vezérlő programok ezen csoportjának alapvető feladata a job-ok végrehajtásának vezérlése. Az ide tartozó rutinok a job-hoz rendelt memória területén elhelyezkedő, privilegizált programok.

A job management első részének, a job monitornak tevékenységét már érintettük a supervisorhoz tartozó tervező müködésének ismertetésekor (ld. 2.1.2.1. fejezet). A job monitor a program megszakitás feldolgozásakor kapja meg a vezérlést, összegyüjti a megszakitásra vonatkozó adatokat, majd meghivja a megfelelő diagnosztikai rutint. Ez a rutin lehet egy, a rendszerhez tartozó hibakezelő program, de lehet a felhasználó által irt program is.

A job management legfontosabb része a munkavezérlő vagy job control nyelv feldolgozója. Ennek tárgyalását a job control nyelvek általános ismertetésével vonjuk össze.

A job control nyelvek az operációs rendszer müködésének irányitására szolgáló vezérlő utasitásokból, vagy direktivákból alakultak ki. E direktivák halmazát néha tévesen job control nyelvnek is hivják.

Az operátor által kiadható direktivák azonban nem teljesitik azt a követelményt, hogy felhasználásukkal a job tulajdonosa megirhassa a job kezelési programját és ezt a programot az operációs rendszer megfelelő része - a job control nyelv feldolgozója - automatikusan értelmezze és végrehajtsa.

Ehhez nem operátori parancsok halmazára, hanem egy tényleges nyelvre van szükség, amelynek segitségével job vezérlő programot lehet irni, ami tulajdonképpen a job-hoz tartozó olyan kezelési utasitás, melyet nem a gépkezelő, hanem közvetlenül az operációs rendszer értelmez.

Milyen előnyökkel jár az ilyen - közvetlenül géppel feldolgozott - kezelési utasitások használata?

Első előny a feldolgozás gyorsasága. A nagy sebességgel müködő számitógép rendszereknél számottevő idűveszteség adódik a manuális operátori beavatkozás következtében, amit a job control nyelv használata jórészt kiküszöböl.

A másik előny a felhasználók rákényszeritése egyértelmü, minden lehetőségre kiterjedő kezelési utasitások - job control programok formájában való - készitésére. A job control nyelv feldolgozója ugyanis hibás, vagy be nem fejezett logikai ágakat tartalmazó job control program esetén nem találja ki - mint a jó operátor - hogy mit kell a job-bal csinálni egy-egy definiálatlan esetben.

Vizsgáljuk meg, hogy milyen feladatokat kell megoldani a job control nyelv segitségével.

Egy job végrehajtásához a következő, lényeges software és hardware erőforrások specifikációjára van szükség. /Irod. 5./:

- a job, valamint a hozzátartozó felhasználói programok és file-ok neveire;

- a szükséges hardware erőforrásokra (ld. 2.) köztük az idő megjelölésére.

Az erőforrások leirásán kivül a job végrehajtásához meg kell adni azokat a feltételeket, amelyeknek teljesülése az egyes jobstep-ek végrehajtásához szükséges. Ezen feltételek leirása a job control nyelvben feltételes, illetve vezérlés-átadási utasitások jelenlétét teszi szükségessé.

Természetesen a fenti funkciókon kivül a job control nyelv még több - itt nem részletezett - funkciót is betölt (pl. kommunikáció az operátorral).

A job control nyelv funkcióit megvalósitó job control programok utasitásai tipus szerint csoportosithatók:

- deklarációs utasitások a job control program elejének és végének, valamint a job nevének stb. megjelölésére;
- program végrehajtást kezdeményező utasitások, programok tárba töltésére és elinditására;
- allokációs utasitások a hardware erőforrásoknak és a file-oknak a job-hoz való hozzárendelésére;
- vezérlő utasitások, amelyek segitségével a job control program utasitásainak végrehajtása feltételessé tehető, valamint
- program tesztelést megkönnyitő utasitások a futtatható formában lévő programok módositására, nyomkövetésére.

A job control nyelv utasitásai a különböző operációs rendszerekben nagy változatosságot mutatnak, ezért valamely létező helyett egy olyan job control utasitás rendszerrel ismerkedünk meg, amely egy fiktiv job control nyelvet alkot és csak a lényegesebb, tipikus elemeket tartalmazza.

Amint erre már utaltunk, a job control nyelvek a vezérlő direktivákon alapulnak és utasitás készlet szintjén - tehát az automatikus feldolgozástól eltekintve - csak a vezérlő utasitások tekintetében tartalmaznak ujat.

A fejlődés során - a programozási nyelvekhez hasonlóan - a job control nyelvek is az általában magas szintünek nevezett nyelvek felé közelednek a felhasználó jobb kiszolgálása érdekében. Ez nem jelenti azt, hogy a magasszintü nyelv fogalma, illetve kritériumai azonosak a programozási és a job control nyelvek esetében, hiszen a két nyelvcsalád különböző igények kielégitésére szolgál.

Ezek alapján gyüjtsük össze azokat az alapvető kritériumokat, melyek egy magasszintü job control nyelvre jellemzők.

- Egyszerüség abban az értelemben, hogy kis gyakorlattal is könnyü legyen hibátlan job control programot irni. Ez a kritérium azért lényeges, mert a job control programok feldolgozási technikájának (ld. később) következtében általában nincs arra lehetőség, hogy egy géphez fordulás folyamán a job control program összes hibái kiderüljenek (ez a programozási nyelvek esetében nem igy van). A legtöbb esetben az első hibás job control utasitás a teljes job végrehajtásának felfüggesztéséhez vezet és a felhasználó csak e hiba kijavitása és a

job elölről való inditása után jut el a job normális befejezéséig (sok esetben csak a következő hibáig).
Az egyszerüség az utasitások viszonylag szabad formájában és a hozzájuk tartozó paraméterekre vonatkozó gyakorlati szempontból ésszerü alapfeltételezések létezésében nyilvánul meg. Ez utóbbi azt jelenti, hogyha egy potenciálisan sok paramétert tartalmazó utasitást a leggyakrabban használt formájában akarunk alkalmazni, akkor ne kelljen a paramétereit kiirni.
Hozzájárul az egyszerüséghez az is, ha az utasitások sorrendje csak indokolt esetben kötött.

- A nyelv vezérlő utasitásokat tartalmaz, melyek segitségével a job végrehajtása során bekövetkező események alapján a job control program utasitásainak végrehajtása feltételessé tehető. Ez a kritérium nagymértékben hozzájárul ahhoz, hogy a job control program olyan feladatot lásson el, amit enélkül a gépkezelő végezne a felhasználó (esetleg nem egyértelmü) utasitásai alapján.
  Ilyen feltételként szerepeltethető esemény lehet pl. egy forditóprogram valamelyik üzenetének megjelenése, ami a forrás program hibás vagy hibátlan voltát jelzi.
- A gyakran használt job control programok, vagy azok egyes részletei makrok formájában irhatók meg, amelyek az assembly nyelveknél szokásos makrokkal azonos módon definiálhatók, paraméterezhetők és használhatók. A makro utasitások hatására a már előzőleg definiált makro az aktuális paraméterek behelyettesitése után a job control program megfelelő helyére másolódik.
  Az assembly makrok-tól a job control makro alkalmazási körülményeiben különbözik. A job control programok alapvető sajátságaiból következik, hogy nincs arra szükség, hogy egy makrot ugyanabban a programban definiáljunk és használjunk, mivel a programok ismétlődő részei nem egy programon belül, hanem általában különböző programokban helyezkednek el. Ebből következően a makrokat egy könyvtárban kell tartani, a könyvtár karbantartása az uj makrok definiálása, könyvtárba való felvétele a job control nyelv segitségével, de a makrokat felhasználó programtól függetlenül történhet.
  A job control nyelvekben a makrok azért játszanak nagy szerepet, mert a job-ok kezeléséhez szükséges legfontosabb tevékenységek nagyon gyakran ismétlődnek. Ilyen pl. a forrásprogramok módositását, forditását és - hibátlan forditás esetén - a tárgyprogram szerkesztését, valamint futtatását elvégző job control program részlet. Egy ilyen makro, melynek paraméterei a szükséges erőforrások (pl. a forditóprogram neve, input-output eszközök, tárigény stb.) nagymértékben csökkentheti a job control programok irásához szükséges mun-

kát és kiküszöböli a program összeállitásakor elkövethető hibák egy részét.

A magasszintü job control nyelvek alapvető tulajdonságainak ismeretében vizsgáljuk meg, milyen tipikus utasitásokból épül fel egy ilyen nyelv.

Az utasitások általános szerkezete a következő:

KULCSSZÓ paraméterek

A kulcsszó, ami általában egy közös karakterrel kezdődik, határozza meg az elvégzendő müveletet, a paraméterek pedig a müvelet végrehajtásához szükséges információt tartalmazzák.

A paraméterek száma széles határok között változik, jóformán mindegyikhez tartozhat alapfeltételezés, ami az utasitásból elhagyott paraméter értéke. Az alapfeltételezések alkalmas megválasztásával elérhető, hogy az utasitások leggyakrabban használt formáiban kell a legkevesebb paramétert megadni.

A job control utasitásokat az előzőekben bevezetett tipusok szerint csoportositjuk, a paramétereket nem definiáljuk pontosan, csak leirjuk, milyen információkat tartalmaznak.

Deklarációs utasitások:

| | |
|---|---|
| JOB | Azonositja a job-ot és a felhasználót a rendszer számára, ezért általában a job control program első utasitása. A job azonositása egy job-névvel, a felhasználóé egy számlaszámmal történhet, ami sok esetben az automatikus gépidő elszámolásnál kerül felhasználásra. |
| END JOB | A job-ra vonatkozó job control program végét jelző utasitás, általában nincs paramétere. |

Program végrehajtást kezdeményező utasitások:

| | |
|---|---|
| LOAD | Egy futtatható formában lévő programot olvas be a központi memóriába. Paraméterként a program nevét és az input eszközt kell megadni. Sok esetben a beolvasás valamelyik külső tárolóban lévő könyvtárból történik, ilyenkor a könyvtár neve azonositja a perifériát. Az utasitás tulajdonképpen allokációs tevékenységet is végez, a kivánt periféria és a szükséges központi memória terület lefoglalásával. Egyes esetekben - különösen az overlay programokat kezelő rendszereknél, a futási idő csökkenthető nagyobb tár hozzárendelésével. Ilyen esetekben az utasitás egy további paramétere a programhoz rendelt memória terület mérete lehet. |

| | |
|---|---|
| UNLOAD | Egy programnak a memóriából való törlésére szolgál. Paramétere a program neve. Az utasitás csak a LOAD-hoz való logikai viszonya alapján szerepel ebben a csoportban. |
| RUN | Egy már beolvasott program elinditása. Paraméterként a program neve, esetleg belépési pont adható meg. Az utasitás paraméter nélkül is használható, ilyenkor az utolsó LOAD által behozott programra vonatkozik. |

Allokációs utasitások:

Az ide tartozó utasitásokat a 2.1.2.3. fejezetben, a Data Management-rutinokkal együtt tárgyaljuk.

Vezérlő utasitások:

| | |
|---|---|
| TIME | Időkorlát a job számára. Paraméterenként a job végrehajtására fordítható idő adható meg. Ennek lejártával a job végrehajtása felfüggesztésre kerül, a job általában csak operátori beavatkozással folytatható. |
| ABORT | A job végrehajtásának felfüggesztése és a job törlése. Az utasitás hatására a paraméterként megnevezett job-hoz tartozó job vezérlő blokk törlődik, a job megszünik létezni a rendszer számára.<br>Az utasitás használatára például akkor van szükség, ha a job valamelyik programja végrehajtás közben olyan állapotba kerül, hogy a job befejezésére nincs lehetőség (végtelen ciklus stb.).<br>Sok esetben a job control programban lévő hibák automatikusan kiváltják az ABORT utasitás végrehajtását, még akkor is, ha az a job control programban nem szerepel. |
| GO TO | Feltétlen vezérlés átadás a job control programon belül. Paraméterként egy cimke szerepel, a job control program végrehajtása a cimkézett utasitással folytatódik. |
| AT | Feltételes utasitás. A job futása közben bekövetkező eseményeket és a rendszer tevékenységeit kapcsolja össze.<br>Általában két paramétere van, egy esemény neve és a hozzátartozó akció neve egy job control |

utasitás formájában. Az esemény igen sokféle lehet a rendszerprogramok sajátságaitól függően. Például:

- időkorlát tullépése
- különféle üzenetek érkezése a rendszer, vagy a felhasználó programjából.

Az utasitás lehetőséget biztosithat több esemény megjelölésére is, melyek közül akármelyiknek a bekövetkezése az utasitásban megadott akciót teszi szükségessé.

Az utasitás hatására egy elágazás jön létre a job control programban.

Ha az első paraméterként szereplő esemény bekövetkezett, a második paraméter által meghatározott job control utasitás kerül végrehajtásra, különben ez az utasitás nem hajtódik végre.

Az utasitás tipikus alkalmazásaként egy forditóprogram hivásából, valamint a tárgyprogram esetleges futtatásából álló job-ot emlithetjük. A forditóprogram müködése után egy AT utasitás segitségével dönthetjük el (a forditóprogramból jövő üzenetek alapján), hogy hibátlan volt-e a forditás és sor kerülhet-e a tárgyprogram betöltésére és inditására.

Program tesztelésére szolgáló utasitások:

| | |
|---|---|
| ALTER | Futtatható formáju, a memóriába betöltött program módositása.<br>Paraméterei egyrészt a módositás helyét (cimét) azonositják, másrészt a megjelölt helyre (szóba, byte-ba) irandó uj értéket tartalmazzák. Egy ALTER utasitással általában több szó (byte) is módositható több paraméterpár segitségével. |
| DUMP | Az operatív memória egy meghatározott területének kiirására szolgál. Sok esetben az automatikusan végrehajtásra kerülő ABORT (ld. ott) utasitáshoz kapcsolódik. |
| TRACE | Egy futó program nyomkövetésére szolgál. A nyomkövetés azt jelenti, hogy a felhasználó részletes információt kap a programjában végrehajtásra kerülő utasitásokról. Általában minden utasitás végrehajtásakor kiirásra kerül maga az utasitás, valamint a gép regisztereinek tartalma (akkumulátor/ok/, cimregiszter, |

| | |
|---|---|
| | indexregiszterek stb.) az utasitás végrehajtása után. Paraméterként a program nyomozandó intervallumát kell megadni két cim formájában. Ha a program végrehajtása közben a megadott intervallumban lévő utasitás kerül végrehajtásra, a nyomozási információk kivitelre kerülnek. Szokás olyan nyomozást is használni, amelyik csak bizonyos utasitásokra (legtöbbször valamilyen vezérlés átadási utasitásra) érzékeny. Nagy jelentősége van a nem gépi kódu, hanem magas szintü nyelveken irott programok nyomozásának. Ebben az esetben általában csak a program speciális üzemmódban (TRACE mód) való ujraforditása után lehet a nyomozást elvégezni.<br>A nyomozandó utasitásokat tipus szerint, vagy a programban szereplő cimkék alapján kell megadni, a nyomozási információ pedig nem a gép regisztereinek, hanem a programhoz tartozó megfelelő változók értékeiből, valamint vezérlési adatokból áll. |
| SNAP | A DUMP és a TRACE utasitások (l. ott) kombinációja. Paraméterként a program egy utasitásának cimét kell megadni. Ezen megjelölt utasitás végrehajtásakor kivitelre kerül a megfelelő TRACE információ, valamint szintén paraméterként megadható módon egy vagy több intervallum DUMP-olása. |
| DISPLAY | Egy üzenet kiirására szolgál a job control program végrehajtása közben. Paraméterként maga az üzenet szerepel. |

A fenti fiktiv job control nyelv feldolgozását a vezérlő nyelv feldolgozó (ld. 1. ábra) végzi. A job control nyelv feldolgozásánál a forditási és értelmezési módszer keveredik, a felhasználó szemszögéből nézve azonban az értelmezési technika dominál, hiszen az egyes job control utasitások feldolgozása tényleges végrehajtásukat is tartalmazza. Sok operációs rendszer alapvető müködési módja a job control utasitásoknak valamelyik input eszközről való egyenkénti beolvasásából és azonnali végrehajtásából áll. Ilyen esetben azt mondhatjuk, hogy a job control nyelv feldolgozása értelmezési technikával történik.

A magas szervezettségü operációs rendszerek azonban több job control program párhuzamos feldolgozását és végrehajtását teszik lehetővé, és itt már szerepet kap a forditási technika is, mivel a végül értelmezésre kerülő job control utasitásokat először valamilyen tömör formáju, az értelmező által könnyen kezelhető alakra kell transzformálni, szintaktikusan ellenőrizni, az esetleges makro utasitásokat kifejteni stb.

### 2. 1. 2. 3. *Data management*

A vezérlő programok utolsó csoportjának feladata az operációs rendszeren belül az adatok kezelése.

A logikailag egymáshoz kapcsolódó adatok legtágabb értelmü halmazát file-nak nevezzük, amely további szorosabban összetartozó adatokat tartalmazó rekordokból épül fel. A rekordok - a file tárolására használt eszköz fizikai tulajdonságainak következtében és az adatátvitel sebességének növelése érdekében - blokkokat alkothatnak.

A számitógépek gyakorlati alkalmazásában az adat-feldolgozási feladatok igen fontos szerepet játszanak, ezért az operációs rendszerek értékének, szinvonalának egyik lényeges mutatója a data management szolgáltatások minősége.

Az adatkezeléssel kapcsolatos tevékenységek alapvető célja, hogy a file-ok kezelését minél jobb hatásfokkal hajtsák végre, és ugyanakkor rugalmasak, jól használhatók legyenek. A file kezeléshez - az input-output müveletekhez - a következő alapvető tevékenységek tartoznak:

- Adatok olvasása
- Adatok irása
- Rekordok blokkositása és blokkok széttördelése
- Az irási, olvasási és a feldolgozási tevékenységek párhuzamositása
- File cimkék kezelése
- File kötetek (volume) automatikus kezelése
- Input-output hibák kezelése.

Ezen célok eléréséhez megfelelő file szervezési, hivatkozási, katalogizáló stb. módszerekre van szükség, melyeket a következőkben röviden áttekintünk.

A *file szervezés* három alapvető változata használatos:

a) *Szekvenciális.* A rekordokat a file-ba fizikai előfordulásuk sorrendjében helyezzük el és ugyanilyen sorrendben történik kiolvasásuk is.

Ilyen file szervezés használatos például mágnesszalagos, lyukkártyás vagy papirszalagos file-ok esetén, de közvetlen hozzáférésü tárolóknál is alkalmazható.

b) *Index-szekvenciális.* Ennél a szervezési módnál a rekordok a file-ban valamilyen *kulcs* értékek szerinti sorrendben helyezkednek el, a kulcs a rekordok részét alkotja. Általában közvetlen hozzáférésü tároló esetén használatos, a rekordok közvetlen módon (kulcs szerint) vagy szekvenciális módon érhetők el.

c) *Direkt*. Közvetlen hozzáférésü tároló eszközön tárolt rekordok közvetlenül elérhetők, ha fizikai cimük (a tárolón való elhelyezést meghatározó adat) ismert. A rekordokat tehát direkt módon irjuk és olvassuk a file-ból.

Az adatok elérési (hozzáférési) módszere, a file szervezésnek megfelelően kétféle lehet: *szekvenciális* hozzáférés és *véletlen* (random) hozzáférés, amikor a rekordokat nem felirási sorrendjükben, hanem kulcs szerint, vagy fizikai tároló cimük szerint érjük el. Szekvenciálisan általában bármilyen szervezésü file hozzáférhető, a véletlen hozzáférés viszont csak nem szekvenciális file-oknál valósitható meg.

A *file-okra való hivatkozás* egy hierarchikus felépitésü névrendszer segitségével valósitható meg. Különböző szintü nevek használatára azért van szükség, mert az operációs rendszeren belül sok file létezhet egy időben, amelyek különböző szempontok szerint (pl. a felhasználó személye alapján) csoportokba sorolhatók. Az egyes csoportok további kisebb csoportokra oszolhatnak és igy tovább. Ennek megfelelően egy konkrét file-ra való hivatkozás például egy A.B.C alaku összetett névvel valósitható meg, ahol A valamilyen - több file-t tartalmazó - csoport neve, B az A csoport egy alcsoportjának neve, végül C a B csoportban egy konkrét file neve.

Az adatstrukturáknak a fentihez hasonló, csak más szemléletü realizálása a *sub-file*-ok bevezetésével is megvalósitható. Ebben az esetben nem file csoportokat, hanem egyetlen, vagy néhány alap, vagy un. *master file*-t és ezen belül sub-file-okat különböztetünk meg. A sub-file-ok ujabb "sub-sub-file"-okat tartalmazhatnak és igy tovább.

A fenti A. B. C hivatkozás ilyenkor az A master file B sub-file-jának C sub-file-ját jelöli.

Ismét más terminológia szerint azt mondhatjuk B és C az A file csoporthoz tartozó *minősitők*, amelyek egyértelmüvé teszik a file-ra való hivatkozást.

Mint rövidesen látni fogjuk, a file-okra való hivatkozások a job control nyelv adatkezeléssel kapcsolatos utasitásaiban szerepelnek, méghozzá nem okvetlenül olyan formában, hogy a hivatkozás egyetlen file-t határoz meg. Például egy három szintü adatstruktura esetén lehetnek olyan müveletek, melyeket nem egyetlen file-on, hanem egy file-csoporton kell végrehajtani, ebben az esetben a hivatkozás A . B alaku lesz.

Az egyes file-ok (esetleg file csoportok) elérése a következőképpen valósitható meg:

a) Minden file-hoz tartozik egy un. *meghatározó-mező*, amely tartalmazza az összes információt, melyre a file-al való bármilyen manipuláció során szükség lehet.

b) Mivel az összetett hivatkozások során egy-egy file-ra sokféle összetett névvel hivatkozhatunk, a magasabb rendü nevek (a file-csoportok nevei) egy lista szerkezetben kerülnek elhelyezésre. Ebben a lista szerkezetben minden minősitő a neki megfelelő eggyel alacsonyabb szintü minősitők listájára mutató információt tartalmaz, végül pedig a legalacsonyabb szintü minősitő a meghatározómezőre mutat.

c) Az a) és b)-ben meghatározott lista szerkezet maga is egy kitüntetett file-t alkot, amit *rendszer-katalógusnak* nevezünk.

A rendszer-katalógus karbantartását (megszüntetett file-ok törlését, uj file-ok bevezetését) a *katalógus kezelő* rutinok végzik. (ld. 1. ábra). A katalógus módositása sok esetben a felhasználó tevékenységének implicit következménye, azonban bizonyos job-control utasitásokkal a felhasználó explicit módon is hozzáférhet a katalógushoz.

A 2.1.2.2. fejezetben szereplő fiktiv job control nyelv data management-tel kapcsolatos fontosabb utasitásai a következők lehetnek:

| | |
|---|---|
| DEFINE DATA | Egy file definiálása jellemző adatainak segitségével. Az utasitás paraméterei tartalmazzák a file-hoz tartozó meghatározó mező kitöltéséhez szükséges információkat. |
| CATALOG | A file adatainak, valamint a rá vonatkozó vezérlő információknak a rendszerkatalógusba való bevezetése. |
| DELETE | Egy file törlése a rendszer katalógusból. |
| DATA | A file-nak tényleges adatokkal való feltöltése. Már előzőleg definiált file-okkal kapcsolatban használható. |
| MODIFY | Egy file módositása.<br>Rekordok beszurását, törlését, vagy kicserélését teszi lehetővé. Müködési módja függ a file szervezésétől. |
| COPY | Egy file másolása. Ez az utasitás sok esetben több kulcsszóval kerül megvalósitásra a különböző másolási lehetőségeknek megfelelően. Például a PRINT kulcsszó a file sornyomtatón való kivitelére, a PUNCH kártyára vagy szalagra való kilyukasztására, a WRITE TAPE a mágnesszalagra irásra szolgálhat. |

A data-management nyujtotta szolgáltatásokat az 1. ábrán látható rutinok a következő megosztásban valósitják meg:

A *hozzáférési módszerek* rutinjai irányitják az átvitelt az input-output berendezések és az operativ tár között. Egy meghatározott input vagy output igény felmerülésekor a vezérlést a megfelelő hozzáférési módszer rutinja kapja (ez egy privilegizált program), amely a felhasználótól kapott információk alapján egy adatblokk olvasását kéri a megfelelő system management rutintól, illetve egy - már összeállitott - blokk kiviteléhez szükséges rutint hivja. Tehát a blokkok rekordokra való bontását (input-nál) illetve a blokkok rekordokból való felépitését (output-nál) a data-management rutinok végzik, de a tényleges fizikai adatmozgatás már a system management rutinok feladata.

A *katalógus kezelés* rutinjai végzik a rendszer katalóguson való manipulációkat, melyek egy része a felhasználó job control utasitásainak végrehajtása közben válik esedékessé. Mivel a katalógus maga is egy közvetlen hozzáférésü tárolón lévő file, a katalógus kezelés rutinjai a hozzáférési módszerek rutinjainak segitségével hajtják végre feladatukat.

Végül az *eszköz kezelés*, valamint a *külső tár kezelés* a megfelelő job control utasitások végrehajtásához szükséges rutin csoport. Az eszköz kezelés a különböző input-output berendezések szétosztásának irányitását végzi és ezen eszközök foglaltsági állapotára jellemző, a teljes rendszerre vonatkozó táblák kezelését tartalmazza.

A külső tár kezelés, egy bizonyos értelemben megkülönböztetett eszköz-csoport, a közvetlen hozzáférésü tárolókkal való gazdálkodást valósitja meg. Alapvető feladata ezen tárolókon való helybiztositás a különböző felhasználók igényei alapján. Az igények például job control utasitások formájában fejeződhetnek ki.

# 3.

# Az operációs rendszerek általános programozási eszközei

Programozástechnikai szempontból vizsgálva, az operációs rendszerek az eszközök igen széles skáláját tartalmazzák. Ezeknek teljességre törekvő felsorolása egyrészt felesleges, másrészt az adott keretek között megoldhatatlan feladat lenne. Azért felesleges, mivel ezeket a programozási eszközöket sok esetben más programokban is alkalmazzák (például táblázatokban való keresés vagy file szervezési módszerek), tehát az operációs rendszerekben való felhasználásuk nem jellemző.

Ezért keressük meg az operációs rendszerek egyik jellemző, általános funkcióját és vizsgáljuk meg ezen funkció megvalósitásához használt eszközöket.

Választásunk az erőforrás szétosztási funkcióra esett (ld. 1.1.) és mivel ez a tevékenység is még nagyon szerteágazó, csak talán legfontosabb részével, a memóriagazdálkodással fogunk részletesebben foglalkozni.

A memóriagazdálkodás azon kivül, hogy tipikus operációs rendszer funkció, központi helyet foglal el a rendszerben, igy a megoldására használatos módszerek áttekintésekor kapcsolatba kerülünk sok más rendszer tevékenység megvalósitására alkalmazott eszközzel is (pl. multiprogramozás).

A következőkben áttekintjük a jelenleg használatos memóriagazdálkodási stratégiákat, illetve algoritmusokat az egyszerübbektől a bonyolultabbak felé haladva.

## *3.1. Supervisor, egy felhasználóval*

Ez a stratégia a legegyszerübb, amely lehetőséget adhat a job-ok automatikus egymásutáni végrehajtására egy batch üzemmódu környezetben.

A supervisor alapfeladata az egyik programról a másikra való automatikus áttérés biztositása, de ellát egyéb tevékenységeket is, mint például a gép- és programhibák kezelése, kapcsolattartás a külvilággal, input-output müveletek vezérlése stb. A hardware-rel szemben támasztott igény minimális, csupán egy védett memória területre van szükség, ahol a supervisor helyezkedik el, az operativ memória többi része korlátlanul elérhető a felhasználó programja által.

A supervisorba való belépés csak valamilyen jól definiált módon (megszakitás, hardware állapot váltás) lehetséges.

A stratégia előnye, hogy az automatikus batch feldolgozást igen jó hatásfokkal valósitja meg. Hátránya, hogy a rendszer átbocsájtó képességét és ezzel együtt a válasz idejét meg sem kisérli optimalizálni.

A válaszidő - azaz egy job leadásától az eredmények megérkezéséig tartó idő - azért lehet hosszu, mert egy sokáig futó job mindenképpen fenntartja az utána érkező rövid job-ot, még akkor is, ha az a várakozási idejének egy töredéke alatt lefutna.

A rendszer hátrányai a tudományos jellegü batch feldolgozásoknál kevésbé szembetünőek, ugyanis az erőforrások megosztott használatának hiánya nem jelent hátrányt olyan programok esetén, amelyek futási idejük tulnyomó részében a központi egységet használják.

### *3.2. Több felhasználó, rögzitett particiókkal*

Ez a stratégia lehetővé teszi az erőforrások megosztott használatát. A memóriában egyidejüleg több felhasználó job-ja van jelen, minden job-hoz egy rögzitett nagyságu memória terület (partició) tartozik.

A 4. ábra egy három particiós memória felosztást tartalmaz.

A müködés alapelve az, hogy valamelyik program addig fut, amig egy input-output utasitást nem akar végrehajtani. Ekkor egy másik - nem input-output közben lévő - program kapja a vezérlést, az input-output müvelet végrehajtásáról pedig az input-output csatornák gondoskodnak függetlenül a központi egységtől.

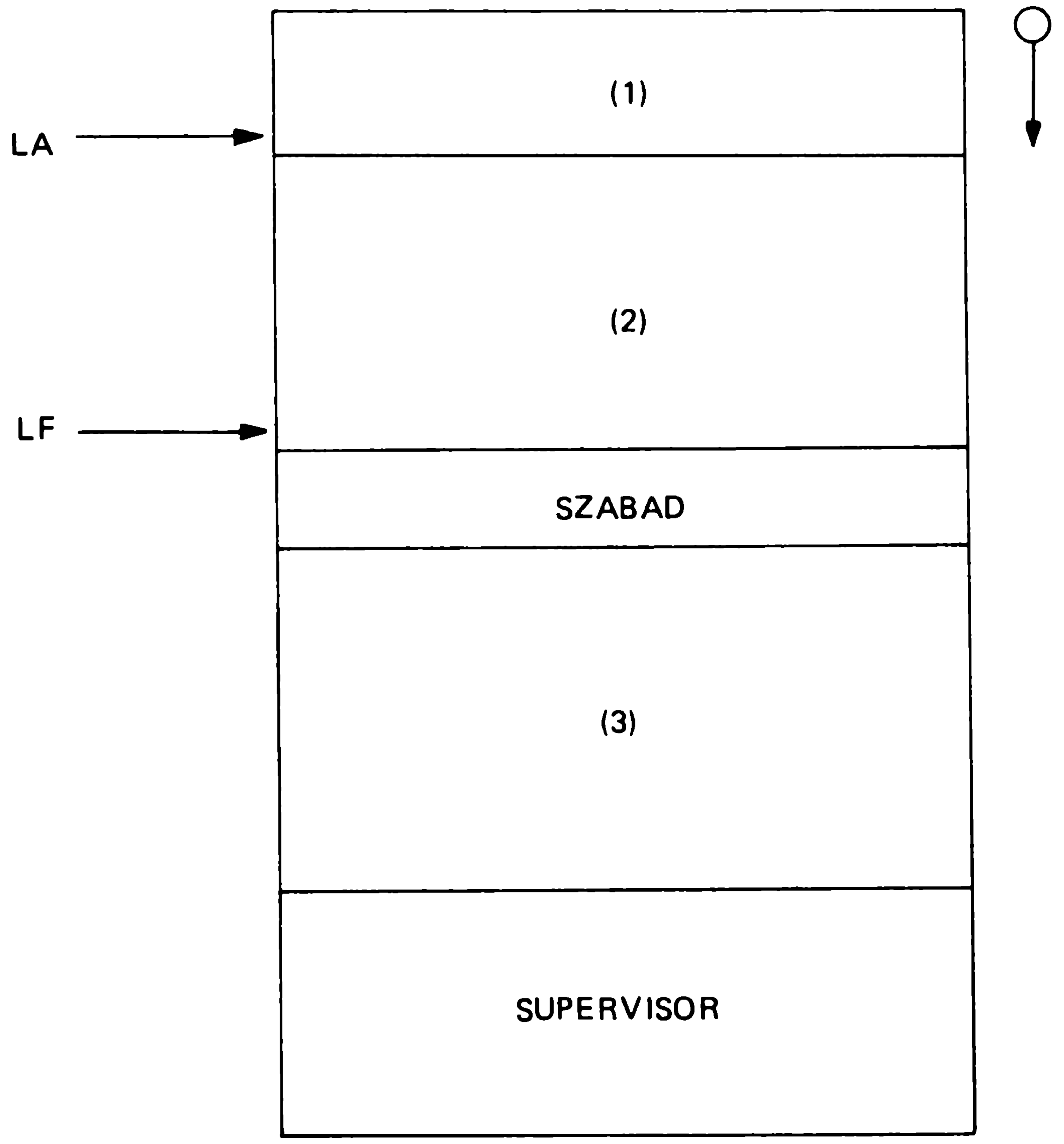
Bármely felhasználói program müködjön is, a saját particióján kivül eső területet védeni kell tőle. Ez pl. két hardware regiszter, az LA és LF segitségével valósitható meg. LA tartalmazza a partició alsó, LF a felső cimhatárát. Minden, a program által elérni szándékozott cimet meg kell vizsgálni, hogy beleesik-e az LA és LF-ben tárolt cimek által meghatározott intervallumba. Ha nem, a vezérlést a supervisor kapja, a hibás program nem folytatható.

Amikor egy másik particióbeli program végrehajtására tér át a rendszer, a supervisor gondoskodik LA és LF uj értékének beállitásáról.

A fentieken kivül célszerü, hogy a rendszer olyan hardware vagy software mechanizmust is tartalmazzon, amely áthelyezhető programok irását teszi lehetővé; igy a felhasználónak nem kell azzal törődnie, hogy melyik particióban fut a programja.

A stratégia előnye az erőforrások megosztott felhasználása, valamint az, hogy minden particióban külön automatikus batch feldolgozás folyhat, tehát a válaszidő csökkenthető, ha a nagyjából azonos ideig tartó job-okat közös particiókban futtatjuk.

A rögzitett particiók hátránya, hogy sok esetben üres hely marad az egyes particiókban, mig olyan nagyobb job-ok nem futtatha-


LA
LF
(1)
(2)
SZABAD
(3)
SUPERVISOR
0

*4. ábra*
*Rögzitett particiók*

tók, amelyek az egy felhasználós rendszerben (ld. 3.1.) lefutottak volna.

Az osztott erőforrás felhasználás különösen adatfeldolgozási feladatok esetében érezteti hatását, amikor gyakoriak az input-output müveletek, azaz rövid idő alatt sok program "jut szóhoz".

A stratégia által megvalósitott multiprogramozás általában prioritásos alapon kerül megvalósitásra. A rendszer olyan környezetben üzemel a leggazdaságosabban, ahol a job-okat idő és tárigény szempontjából néhány nagyobb csoportba lehet sorolni.

### 3.3. *Több felhasználó változó particiókkal*

A 3.2.-ben bemutatott tárkezelési technika egy fejlesztett változata ez a stratégia, amely rugalmas és jobb hatásfoku tárkihasználást biztosit. A központi memória particiókra osztása és az egyes job-ok multiprogramozott végrehajtása megmarad. Az ujdonságot a supervisor azon képessége jelenti, hogy a particiók méretét és számát a feldolgozás alatt álló job-ok nagyságához és számához igazitja.

Az 5. ábra két kisebb és egy nagyobb job-ot ábrázol (a). A (2) jelü job törlésekor a két megmaradó program ugy helyeződik át, hogy ujra a tár elejétől kezdődő folyamatos részt foglaljon el (b). A felszabadult helyen ujabb job-ok végrehajtása kezdődhet (c).

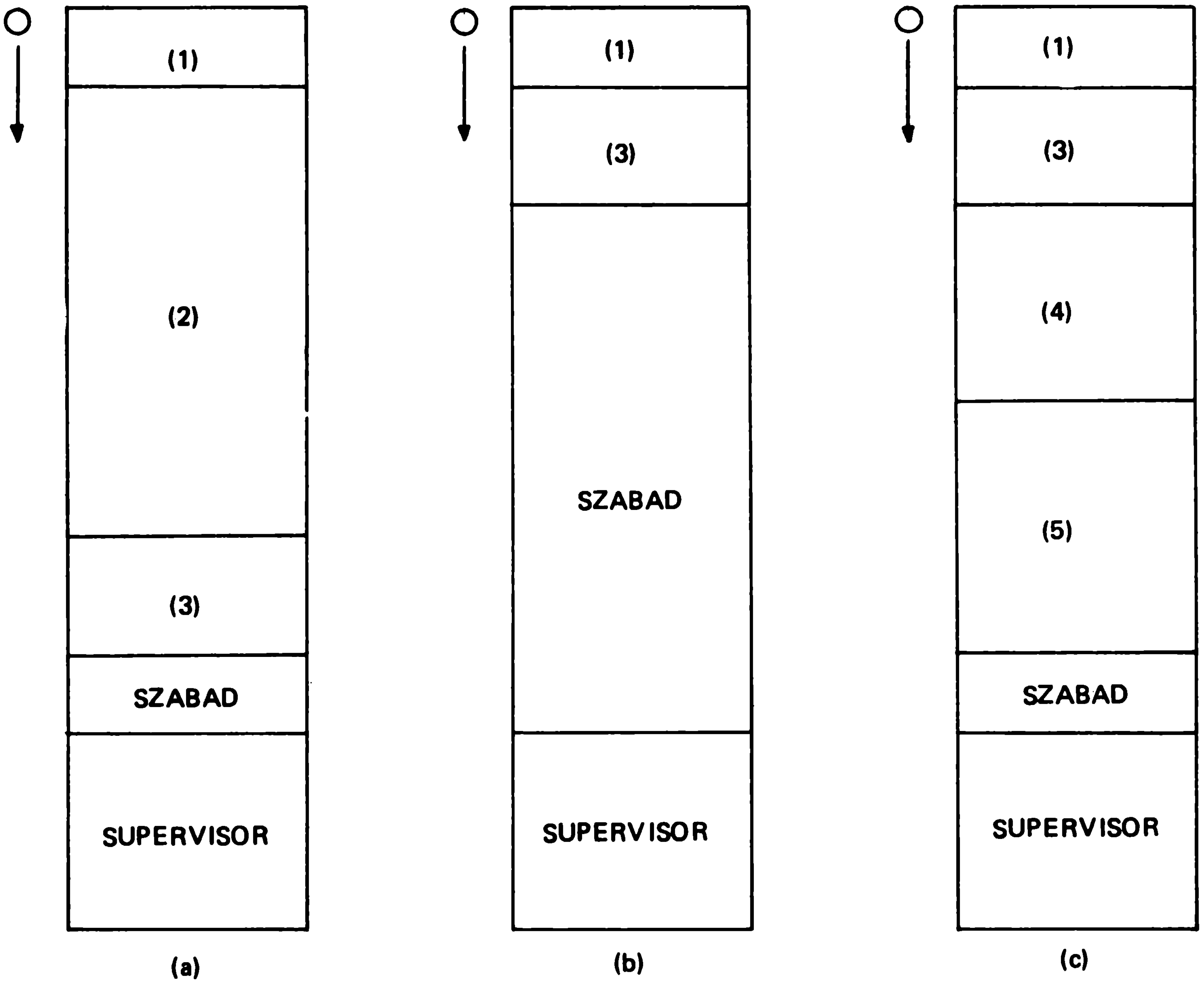
Mint a rögzitett particiók esetén, itt is szükség van hardware memória védelemre. Ezenkivül kell még megvalósitani a programok hardware utján való áthelyezését is.

A stratégia előnye rugalmasságában rejlik: a supervisor a pillanatnyi terhelésnek megfelelően szabályozza a particiók számát és nagyságát. Lehet sok kis, vagy kevesebb nagy méretü particiónk, és lehetőség van igen nagy memória igényü job-ok kezelésére is - esetleg egyetlen partició keretében - amit a rögzitett particiók mellett nem tudnánk feldolgozni. Általános esetben több program van a memóriában, tehát az erőforrások megosztott felhasználása jól érvényesül.

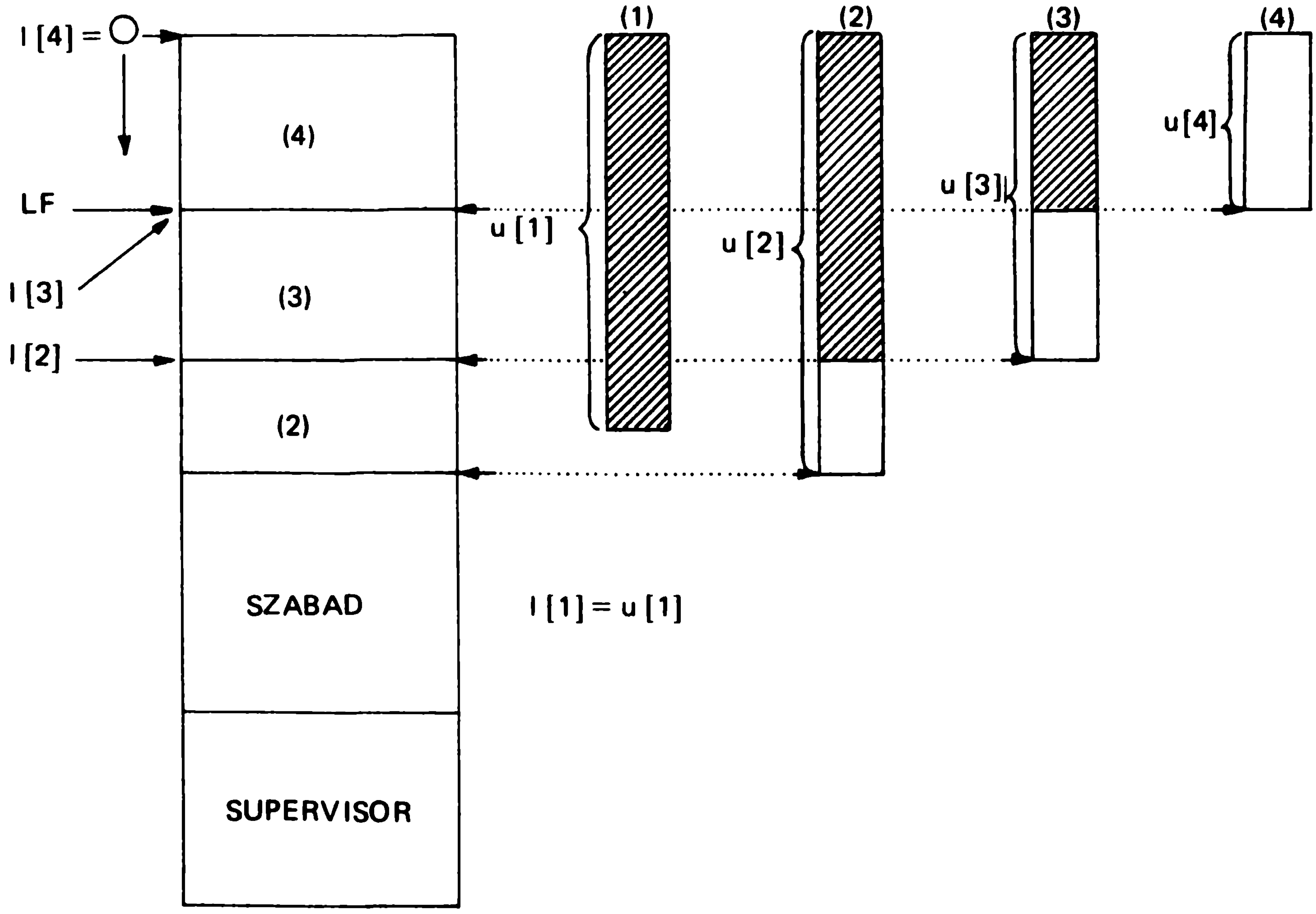
A stratégia fő hátránya, hogy a programoknak a tárban való áthelyezése időt vesz igénybe. Ezt az időt növeli az a tény is, hogy a supervisornak biztositania kell, hogy az áthelyezésre kerülő programokban ne folyjon input-output tevékenység, azaz meg kell várnia ezek befejeződését, mielőtt az áthelyezés megtörténne.

Összefoglalva: a rugalmasságért bonyolultabb tárkezelési algoritmussal kell fizetnünk.

A rendszer jól beválik közepes és nagy teljesitményü gépek batch üzemmódu alkalmazásánál. Hátrányai abban az esetben mutatkoznak, ha az üzemmód következtében a job-ok nagyon gyakran cserélődnek, viszonylag kis időt töltve az operativ memóriában (pl. un. multi-access környezet esetén).


0
(1)
(2)
(3)
SZABAD
SUPERVISOR
(a)
0
(1)
(3)
SZABAD
SUPERVISOR
(b)
0
(1)
(3)
(4)
(5)
SZABAD
SUPERVISOR
(c)

*5. ábra*
*Változó partíciók*


l [4] = O
LF
l [3]
l [2]
(4)
(3)
(2)
SZABAD
SUPERVISOR
(1)
(2)
(3)
(4)
u [1]
u [2]
u [3]
u[4]
l [1] = u [1]

*6. ábra*
*Hagymahéj algoritmus*

## *3.4. A hagymahéj algoritmus*

Az eddig bemutatott tárkezelési technikák kizárólag az operativ memória felosztására vonatkoznak, a háttér memória kezelése hallgatólagosan explicit input-output müveletek segitségével történik. Az ugynevezett hagymahéj algoritmus és az ezután következő módszerek a háttérmemória automatikus kezelésével igyekeznek javitani a központi memória kihasználtságán és növelni a rendszer átbocsájtóképességét.

A hagymahéj elnevezés arra utal, hogy a módszer - bár a több particiós stratégiákhoz áll közel - az egyes particiók számára nem közös rész nélküli memóriarészeket tart fenn, hanem azokat egymásra fektetve kezeli.

A megoldás lényege az, hogy a pillanatnyilag futó program minden szükséges tartozékával együtt az operativ memóriában van, az összes többi program (amelyek pillanatnyilag nem futnak) háttérmemóriában kerül tárolásra. Az egyik programról a másikra való áttérés a multiprogramozás algoritmusa szerint következik be és ilyenkor a megfelelő program automatikusan másolódik be az operativ tárba, részben vagy egészben felülirva az előzőleg aktiv programot, amely viszont minden járulékos információval (regiszterek tartalma stb.) együtt kimásolódott a háttértárba.

Az algoritmus igyekszik a háttértárral való információ forgalmat minimalizálni oly módon, hogy ha egy hosszabb program helyére egy rövidebb jön be, a hosszabb program végét nem viszi ki és nem hozza vissza, hanem megtartja az operativ memóriában.

A 6. ábra négy programot ábrázol különböző végrehajtási állapotokban.

A programok tárigényét a megfelelő keskeny téglalapok szemléltetik. A téglalapok vonalkázott része a háttérmemóriában lévő programdarabot, fehér része az operativ tárbeli darabot jelzi.

A már emlitett LF regiszter védi a memóriának az éppen futó programon kivüli részét.

Pillanatnyilag éppen a (4) jelü program aktiv. Mivel ez a legrövidebb, lehetőség van arra, hogy az előzőleg futott (3) és (2) jelü programok egy-egy darabja is az operativ memóriában várja, hogy sorra kerüljön.

Az operativ memória valóban egy hagymához hasonlit, melynek belső része az éppen futó program, az egymásra simuló héjai pedig az előzőleg aktiv programok maradékai.

Az algoritmus lényegét leiró ALGOL-60 program a deklarációk mellőzésével a következő:

Jelölések:

N: a kezelhető programok maximális száma

u [i] : $0 \le i \le N-1$ esetén az i. program által igényelt tárterület (szavakban, v. byte-okban); u [i] = 0, ha az i. program nem létezik.

| | |
|---|---|
| l [i]: | 0 ≤ i ≤ N-1 esetén a legkisebb cim, amit az i. program az operativ memóriában elfoglal.<br>l [i] = u [i] ha a program teljes egészében csak a háttértárban van. |
| x [i]: | Az i. program számára fenntartott háttér memóriabeli terület kezdőcime. |
| output (a, b-1) to: (z): | Egy kiviteli eljárás, a-tól számitott b db szót visz ki a háttértár z cimétől kezdődő helyre. |
| input (a, b-1) from: (z): | Input eljárás, a háttértár z cimétől kezdve b szót hoz be az a-nál kezdődő helyre. |

A program először gondoskodik a pillanatnyi memória tartalom felülirásra kerülő részének kimentéséről, majd beolvassa az i. programot.

```
for j:=0 step 1 until N-1 do
  if l [j]   l [i] then
    begin m:= min (l [i], u [j]);
          output (l [j], m-1) to: (x [j] + l [j]);
            l [j] :=m
    end helycsinálás;
  input (0, l [i] -1) from: (x [i] );
  l [i] :=0;
  LF:= u [i]
```

A módszer megvalósitásához szükséges hardware csupán az LF regiszter megfelelő kezeléséből áll. A nyilvánvaló előnyök (kis programok jó hatásfoku kezelése, maximális nagyságu programok kezelése) mellett gyenge pontok a következők:

- Csökkenti a megosztott erőforrás felhasználás mértékét, mivel minden programváltás háttérmemóriához fordulást igényel, tehát nem érdemes programot cserélni normál perifériára való várakozás esetén. Indokolt azonban a programváltás az operátorra való várakozásnál (pl. interaktiv üzemmódu konzol irógépre való várás esetén).
- A program váltáshoz szükséges input-output tevékenység nem lapolható át más hasznos munkával, mivel ilyenkor egyetlen program sincs teljes egészében az operativ memóriában.

- Általában az operativ memória egy darabja állandóan kihasználatlan (ha a programok között egy sincs, amelyik a teljes rendelkezésre álló operativ memóriát betöltené).

A hagymahéj algoritmus legjobban közepes méretü rendszereknél válik be, ahol a munka nagyon változó méretü job-okból áll, melyek azonos tár és futási idő paraméterek szerint nem csoportosithatók.

## *3.5. Minimális overlay módszer*

Ez a stratégia a hagymahéj algoritmus módositása abban az értelemben, hogy nem feltételezi azt, hogy az egyes programok részére lefoglalt területek az operativ memória elején kezdődnek. Bármely program a memóriában tetszőleges helyen kezdődő, megfelelő méretü folytonos területet foglalhat el, amely ugy kerül kiválasztásra, hogy a többszörösen lefedett terület nagysága - és ezzel együtt a program váltásnál kimentendő és behozandó információ mennyisége minimális legyen.

A 7. ábra a 6. ábrán mutatott négy program végrehajtása közben kialakult memória térképet tartalmazza, azonos jelölésekkel.

Először az (1) program indult, ami a tár elején kapott helyet. Amikor a (2) program került betöltésre - az overlay terület minimalizálása érdekében - helye a tár felső részén helyezkedett el, az (1) program egy kis darabját azonban még igy is ki kellett menteni. A (3) program inditásakor a tárban lévő két program egyikének nagy részét menteni kellett, az algoritmus azonban biztositotta, hogy a már kétszeresen lefedett tárrész nem került ujra kiosztásra. Hasonló okból került a (4) program a tár elejére, és - szerencsés módon - együtt tartható bent a (3) programmal.

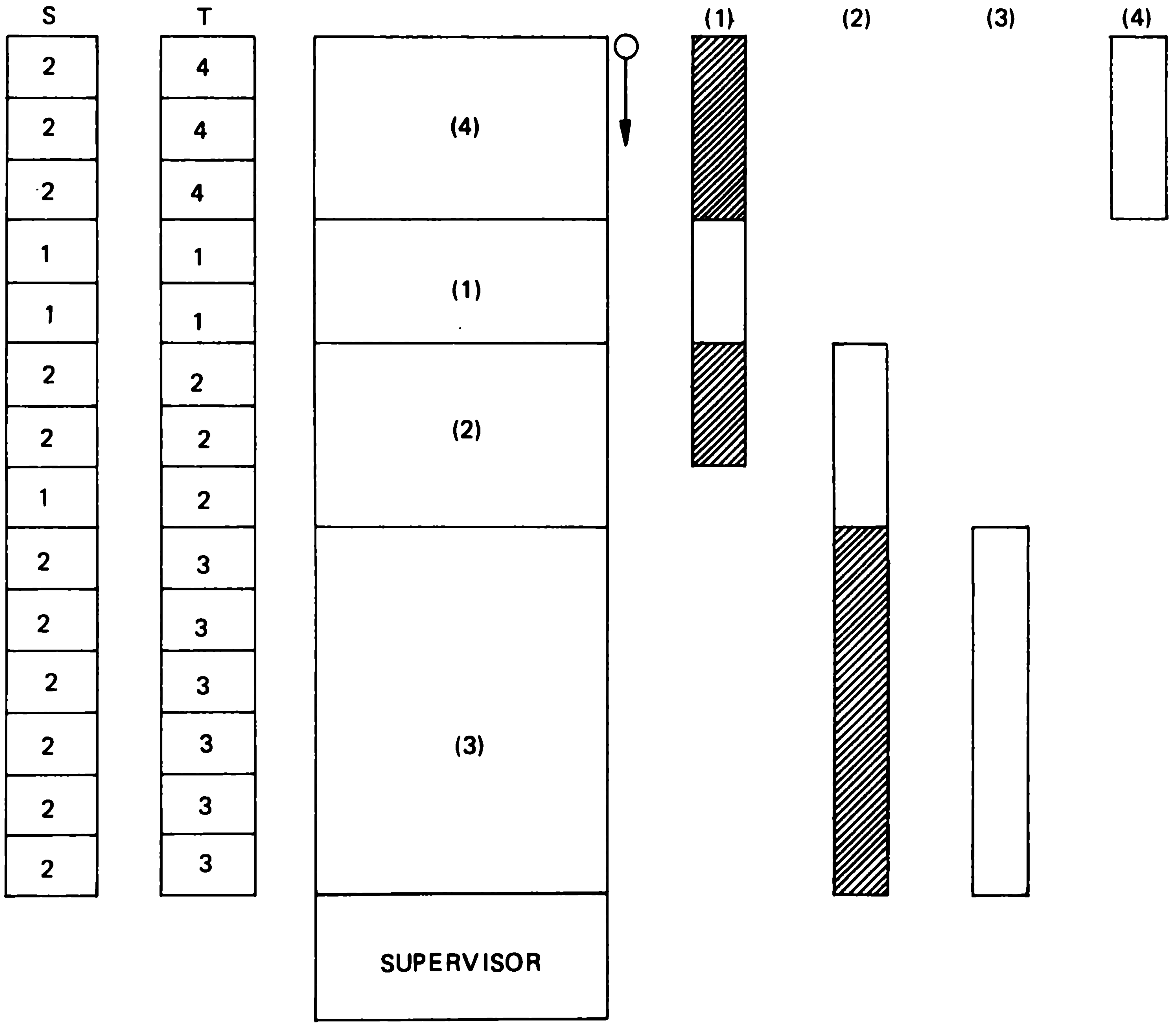
Az algoritmus technikai megvalósitásához az operativ memóriát kis, azonos méretü egységekre, ugynevezett lapokra kell osztani (tipikus lapméret például 1024 szó). Minden programhoz egész számu lap tartozik.

Szükségünk van továbbá két táblázatra is:

- az S overlay számláló minden laphoz megadja, hogy hány programhoz van pillanatnyilag rendelve,
- a T tulajdonos tábla minden laphoz megadja azon program azonositó számát, amely a lapot pillanatnyilag az operativ memóriában elfoglalja (lásd a 7. ábrát).

  Az i. program futtatásának előkészitése abból áll, hogy a program által igényelt lapokat végig kell nézni a tulajdonos szempontjából. Ha egy ilyen lap pillanatnyi tulajdonosa nem az i. program, a lap tartalmát pillanatnyi tulajdonosához tartozó háttérbeli területre kell menteni és uj tartalmát a háttértár i. programhoz tartozó részéről feltölteni.

  Az erre szolgáló ALGOL-60 program a deklarációktól eltekintve a következő:


S
2
2
2
1
1
2
2
1
2
2
2
2
2
2
T
4
4
4
1
1
2
2
2
3
3
3
3
3
3
(4)
(1)
(2)
(3)
SUPERVISOR
(1)
(2)
(3)
(4)

*7. ábra*
*Minimális overlay*

Jelölések:

| | |
|---|---|
| N: | a kezelhető programok maximális száma |
| l [i] : | $0 \leq i \leq N-1$ esetén az i. programhoz rendelt legkisebb cim az operativ memóriában. |
| u [i] : | $0 \leq i \leq N$ esetén az i. programhoz rendelt legnagyobb cim az operativ memóriában. |
| x [i] : | Az i. program számára fenntartott háttér memóriabeli terület kezdőcime. |
| P: | A lapméret. |

```
for j:= l [i] step P until u [i] do
    if T [j÷P] ≠ i then
       begin output (j, j+P-1) to:
          (x [T [j÷P]] + j-1 [T [j÷P]]);
             input (j, j+P-1) from:
               (x [i] +j-1 [i]);
       end;
    LA:= l [i];  LF:= u [i]
```

A bonyolultabb feladat az i. program helyét kiválasztani az operativ memóriában. Ez a feladat a gyakorlatban megelőzi a fenti betöltő algoritmus működését és egy uj fogalomnak, a "programok sulyának" bevezetését teszi szükségessé.

Az előzőleg definiált S tábla tartalmát módositsuk ugy, hogy a tábla i. eleme ne azt mutassa, hogy az i. lap hányszorosan van lefedve, hanem tartalmazza azon programok sulyainak összegét, melyekhez az i. lap pillanatnyilag tartozik.

Ilyen módon egy program sulya - egy pozitiv egész szám - azt fejezi ki, hogy mennyire fontos az ezen programhoz tartozó lapokat az operativ memóriában tartani. A minimális overlay módszer annál ritkábban fogja az i. lapot az operativ tárban kicserélni, minél nagyobb az S [i] értéke.

A programok sulyozására abból a gyakorlati szempontból van szükség, hogy egyes alapvető fontosságu rutinokat (pl. loaderek, forditó programok stb.) csak végső esetben célszerü az operativ tárból eltávolitani.

A következő algoritmus minden lehetséges elhelyezéshez meghatározza az érintett S táblabeli elemek összegét és ezek közül a legkisebbet választván, adja meg az i. program elhelyezési paramétereit és az S tábla megfelelő elemeinek uj értékeit.

Az algoritmus ALGOL-60 programja a következő:

Jelölések, az előző algoritmus jelölésein kivül:

SL: az operativ memória mérete lapokban

L: az i. program mérete lapokban

W: az i. program sulya.

```
M:= 0;
for k:= 0 step 1 until L-1 do M:=M+S [k];
POS:= L-1; MIN:= M;
for j:= L step 1 until SL-1 do
    begin M:=M-S [j-L] + S [j];
     if M < MIN then
        begin POS := j; MIN:= M end
    end;
l [i] := P*(POS-L+1);
u [i] := P*POS+P-1;
for j:= POS-L+1 step 1 until POS do
      S [j] := S [j] + W;
```

Természetesen lapcsere esetén a T táblázatot is fel kell ujitani.

A módszerhez szükséges speciális hardware a 3.2.-ben bevezetett LA és LF regiszterek kezelését valósitja meg. Természetes követelmény a programok áthelyezhetőségének megoldása is, amely akár hardware, akár software uton megoldható.

A módszer előnyei a következőkben foglalhatók össze:

- lehetőséget nyujt a megosztott erőforrás használatra, ha elegendően nagy operativ memória áll rendelkezésre. Ebből a célból a periféria kezelő programoknak nagy sulyt célszerü adni, hogy nagy eséllyel állandóan az operativ memóriában maradhassanak.
- Lehetőség van a program csere idején egy másik program végrehajtására.

Foglaljuk végül össze a módszer hátrányait is (ezek tulajdonképpen az összes előző módszer hátrányaiként is felhozhatók):

- A futó programnak teljes egészében az operativ tárban kell lenni.
- Semmilyen tárolt információ nem tartozhat egyszerre több programhoz. Emiatt az általános használatu rutinokat (pl. standard függvények programjait) minden program külön tartalmazza és viszi, illetve hozza magával programcsere esetén. Csak a supervisor rutinjait használhatja több program közösen.

A módszer legelőnyösebb alkalmazási területe nagyjából egybeesik a hagymahéj algoritmuséval.

## *3.6. Késleltetett betöltés*

A most ismertetésre kerülő módszer kiküszöböli az eddig bemutatott stratégiák (ld. 3.1.-3.5. fejezet) azon hátrányát, hogy a programok csak akkor futhattak, ha teljes egészükben az operativ memóriában helyezkedtek el.

Az operativ memória most is lapokra oszlik, de az egyes programokhoz rendelt lapok csak akkor kerülnek beolvasásra a háttérmemóriából, ha a futó program ténylegesen igényli azokat.

A 8. ábrán a (4) program eddig csak az első lapját használta, ugyanakkor a (3) program már egy kivételével minden lapját igényelte.

A módszer megvalósitásához szükség van a 3.5.-ben szereplő T táblára, amely minden laphoz megadja, hogy pillanatnyilag melyik programhoz tartozik.

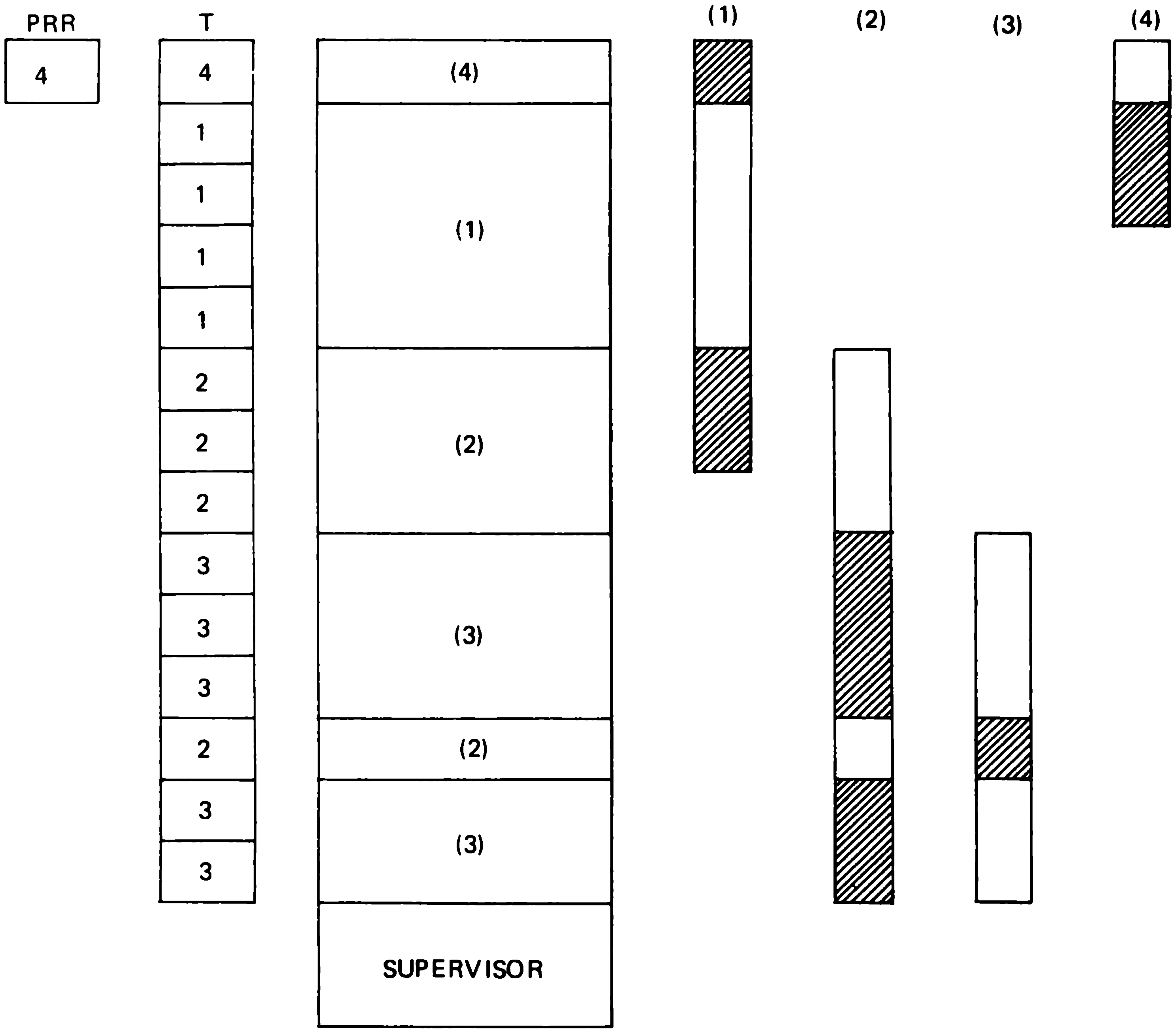
A futásra kiválasztott programhoz a supervisor rendeli hozzá az igényelt területet és beállit egy ujabb speciális regisztert a PRR (ld. 8. ábra) nevüt, amely a pillanatnyilag futó program azonositóját tartalmazza. A program ezután ugy kezd el futni, hogy csak az első lapja van az operativ memóriában. A futás során generált összes cimre a hardware összehasonlitja a T tábla megfelelő elemét PRR tartalmával. Ha azonosak, a kérdéses utasitás végrehajtható. Ha különböznek, egy megszakitás eredményeként a supervisor kapja a vezérlést, amely gondoskodik az igényelt lapnak az operativ tárba való behozataláról és a program folytatásáról.

Mielőtt egy uj lap beolvasásra kerülne, a supervisor eldönti a kérdéses területet pillanatnyilag elfoglaló lap sorsát.

Három lehetőség van:

- Egyik program sem használja a kérdéses lap területet. (Ilyenkor az uj lap azonnal beolvasható.)
- A területet egy másik program foglalja el és a lapon tárolt információ a beolvasása óta megváltozott.
  Ebben az esetben a kérdéses lapot vissza kell másolni a megfelelő helyre a háttér memóriába az uj lap behozatala előtt.
- A területet egy másik program foglalja el, de a lapon tárolt információ a beolvasása óta nem változott. Ilyenkor a terület kimentés nélkül felülirható, hiszen egy másolat a háttérmemóriában megtalálható.

A második és harmadik eset szétválasztása egy speciális hardware regiszter segitségével történik. Minden laphoz egy "irás-bit" tartozik, amely a lap beolvasásakor törlődik. Amikor a lapon belül bármelyik szó tartalma megváltozik, az irás-bit értéke 1 lesz, jelezvén a supervisor számára, hogy a lap tartalmát felülirás előtt ki kell menteni.


PRR
4
T
4
1
1
1
1
2
2
2
3
3
3
2
3
3
(4)
(1)
(2)
(3)
(2)
(3)
SUPERVISOR
(1)
(2)
(3)
(4)

*8. ábra*
*Késleltetett betöltés*

Ha ilyen irás-bit nincs, a harmadik esetet is ugy kell kezelni, mint a másodikat, ami természetesen idővesztességet eredményez.

A módszer jelentős előnye, hogy sok esetben számottevő mértékben csökkenti az operativ memória és a háttértár közötti adatforgalmat. Általában valamely más tárkezelési algoritmussal (legtöbbször a minimális overlay módszerrel) kombinálva használatos, amely növeli annak valószinüségét, hogy egy adott program által használt lapra nem fog egy másik program is igényt támasztani.

Különösen hasznosnak bizonyult a módszer olyan környezetben, ahol a munka nagy részét programok tesztelése alkotja, ugyanis ezek a programok általában hamar müködésképtelenné válnak, mielőtt számottevő részük végrehajtására sor került volna.

A módszer fő hátránya, hogy nem nyujt védelmet olyan esetek ellen, amikor egy lapot két program nagy gyakorisággal használ, ennek következtében a lap tartalma állandóan cserélődik, holott bőven lenne hely a lapon tárolt cserélődő információ párhuzamos tárolására. Ennek egy lehetséges megoldása a lapcserék gyakoriságának korlátozása, ami azonban nem oldja meg a megnövekedett hardware igényekből származó, sebesség csökkenésben és a supervisor bonyolultságában jelentkező hátrányokat.

## 3.7. *Lapkezelés (Paging)*

A tár kezelési stratégiák tárgyalásában az egyszerü módszerektől a bonyolultabbak felé haladva elérkeztünk az utolsóhoz, a lapkezeléshez.

A lap fogalma nem uj, már a minimális overlay módszernél (ld. 3.5.) bevezettük, abból a célból, hogy megkönnyitsük az operativ tárral való gazdálkodást. Az eddig ismertetett módszerek azonban nem aknázták ki azokat a lehetőségeket, melyek a memória lapokra osztásában rejlenek.

Mielőtt a lapkezelés általános elméletére és a hozzátartozó algoritmusok ismertetésére rátérnénk, vizsgáljuk meg, hogy a 3.1.-3.6.-ban bemutatott módszerek milyen kivánnivalókat hagytak maguk után, azaz mely problémák megoldását várjuk a lapkezelés általánositott módszerétől.

- Az első alapvető hiányosságot az okozta, hogy a háttértárból beolvasott lapnak az operativ memória egy rögzitett helyére kellett kerülnie, hogy a futó program által helyesen cimezhető legyen. Ha ezt a területet történetesen egy másik program is nagy gyakorisággal használta, akkor a háttérmemóriával való információcsere intenzitása a hasznos munka lassulását eredményezte.
- A második megoldatlan kérdés a programok vagy adatok közös használata volt. Nyilvánvalóan gazdaságosabb a több program által használt rutinokat (vagy adatokat) csak egyetlen pél-

dányban tárolni és biztositani megosztott felhasználásukat a többi erőforrásához hasonlóan.

- Végül egyik módszer sem próbált megoldást adni olyan programok kezelésére, melyek egyszerre nem férnek el az operativ memóriában. Ilyen nagy programok esetén a programozó szervezte meg programjának (és adatainak) a háttértárból az operativ memóriába való szegmensenkénti behozatalát. Ez nemcsak kényelmetlenséget jelent a felhasználó számára, hanem olyan feladat elé állitja, amit gazdaságosan megoldani nem is képes. Mivel a programozó által előirt adatátviteli müveletek az operációs rendszer memória kezelő stratégiájával keveredve kerülnek végrehajtásra, a felhasználó nem tudhatja, hogy sok kicsi, vagy kevés nagy overlay szegmenssel dolgozzon-e, hogy a programja végül is gyorsan lefusson. Erre a kérdésre csak a többi párhuzamosan futó program szerkezetének ismeretében lehet válaszolni, ez pedig már az operációs rendszer, nem pedig az egyéni felhasználók hatáskörébe tartozik.

Az első probléma megoldása az, hogy egy beolvasott lap az operativ memóriában akárhol elhelyezhető, ahol szabad hely van. Ebben az esetben egy programhoz tartozó lapok az operativ memóriában szétszórva helyezkedhetnek el, a program számára ismeretlen cimeken.

Ebből következően a programban szereplő cimek nem valódiak, hanem virtuálisak, azokat felhasználásuk előtt tényleges cimekké kell konvertálni. Ez a konverzió már nyilván nem oldható meg egyetlen LA regiszter tartalmának a cimhez való hozzáadásával.

A cim konverziót gyakorlatilag ugy oldhatjuk meg, hogy egy táblázatot épitünk fel, melyben a programhoz tartozó minden laphoz egy-egy LA regisztert tárolunk. Ha egy lap pillanatnyilag az operativ memóriában van, megadja a háttértárbeli cimét (természetesen egy jelző bitre is szükség van, amely a két esetet különválasztja). Ezt a táblázatot "Lap-táblá"-nak nevezzük, alkalmazását a 9. ábra illusztrálja.

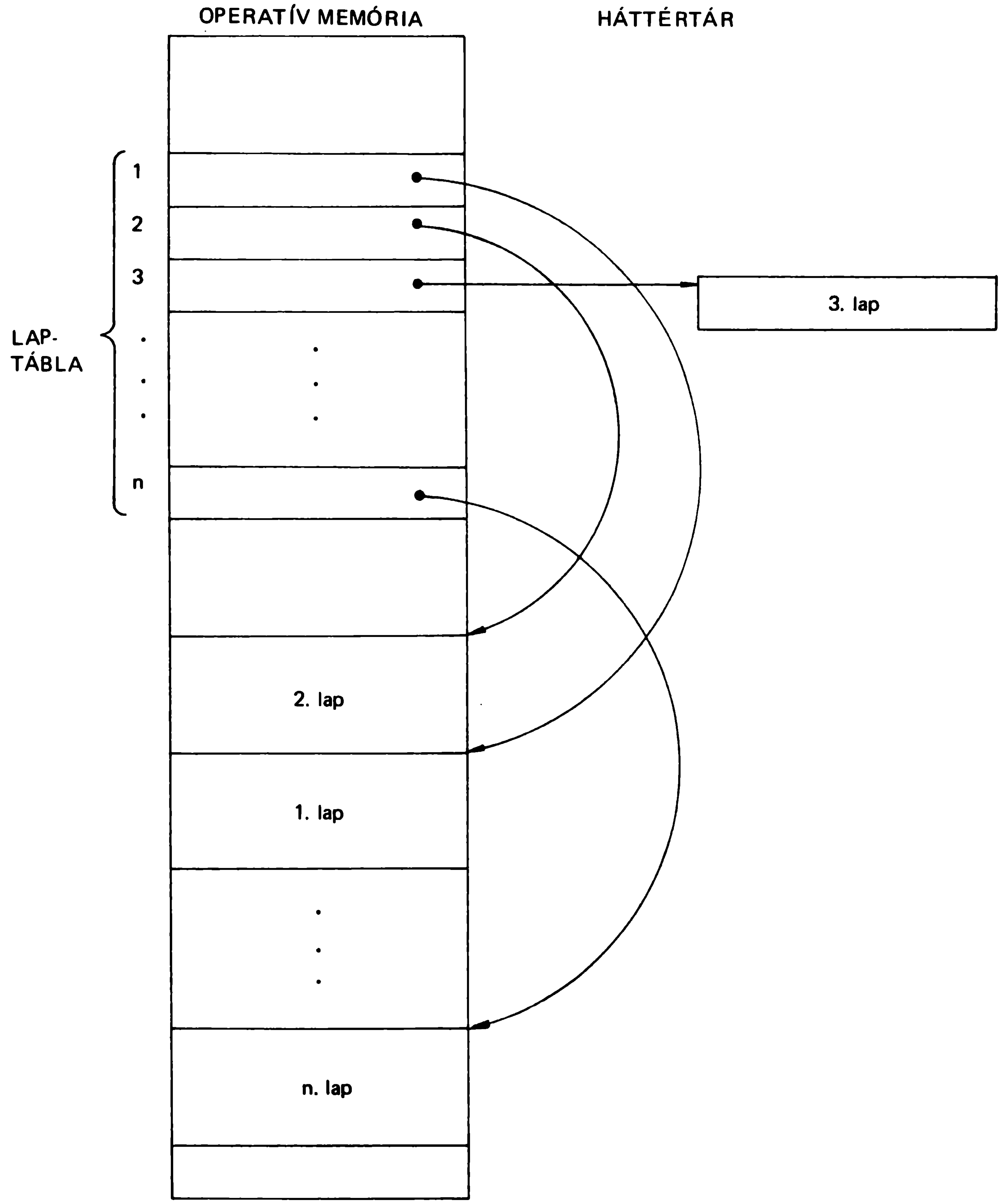
A lap-tábla felhasználásával a cimkonverzió a következőképpen végezhető el:

A program végrehajtása közben egy hardware regiszter a programhoz tartozó lap-tábla kezdetére mutat. A programban használt virtuális cimek két részből tevődnek össze:

- a cim felső része adja a lap-cimet,
- az alsó része pedig a lapon belüli szó-cimet.

(Ez a felbontás feltételezi, hogy a lapméret teljes kettő-hatvány.)

A lap-cimet a lap-tábla indexeként használva abból kiválaszthatjuk a megfelelő elemet, amit lap-mutatónak nevezünk. A lapmutató egy jelző bitet és egy kezdőcimet tartalmaz.


OPERATÍV MEMÓRIA
HÁTTÉRTÁR
LAP-
TÁBLA
1
2
3
.
.
.
n
3. lap
2. lap
1. lap
n. lap

*9. ábra*
*Lap-tábla*

A jelzőbit megmondja, hogy a lap az operativ memóriában van-e. Ha igen, a kezdőcim és az eredeti szó-cim összege adja a kivánt operativ memóriabeli cimet.

Ha a jelzőbit azt mutatja, hogy a kivánt lap a háttértárban van, egy megszakitás utján a megfelelő supervisor rutin gondoskodik behozataláról még a cimhez való hozzáférés előtt.

A 10. ábra a cimkonverzió folyamatát illusztrálja.

Az eredetileg lapkezelésre tervezett gépeknél (un. paged computers) a fent leirt cimkonverzió hardware eszközök segitségével kerül megvalósitásra az operativ memóriához fordulás elválaszthatatlan részeként.

A cimkonverzió software eszközökkel is megvalósitható, de - mint láttuk - minden cim elérése két tárhoz fordulásból áll: első a lap-mutató kiválasztása, a második a kivánt szó elérése. Ezen folyamat felgyorsitására szolgálhat néhány (8-16) igen gyors elérésü asszociativ regiszter, melyek egy un. asszociativ tárat alkotnak.

Mindegyik asszociativ regiszter két mezőből áll, az első - asszociativ - mező egy, az operativ memóriában lévő lap lapcime, a második a lap memóriabeli kezdőcime.

A virtuális cim lapcim része szimultán módon kerül összehasonlitásra a regiszterek asszociativ mezőivel. Mivel az asszociativ tárban olyan lap-mutatók vannak, melyek az utolsó memóriához fordulásoknál kerültek kiválasztásra, a szimultán összehasonlitásnál nagy az egyezés valószinüsége, azaz megvan a kivánt lap tényleges kezdőcime.

Ha nem volt egyezés, a keresett lap-mutatót a lap-táblából kell kikeresni és átirni az asszociativ tárba - általában az ott legrégebben tartózkodó lap-mutató helyére.

Amikor egyik program végrehajtásáról egy másikra térünk át, az asszociativ tárat törölni kell, mert az uj program egy egészen más lap halmazra akar majd hivatkozni esetleg ugyanazon lap-cimek segitségével.

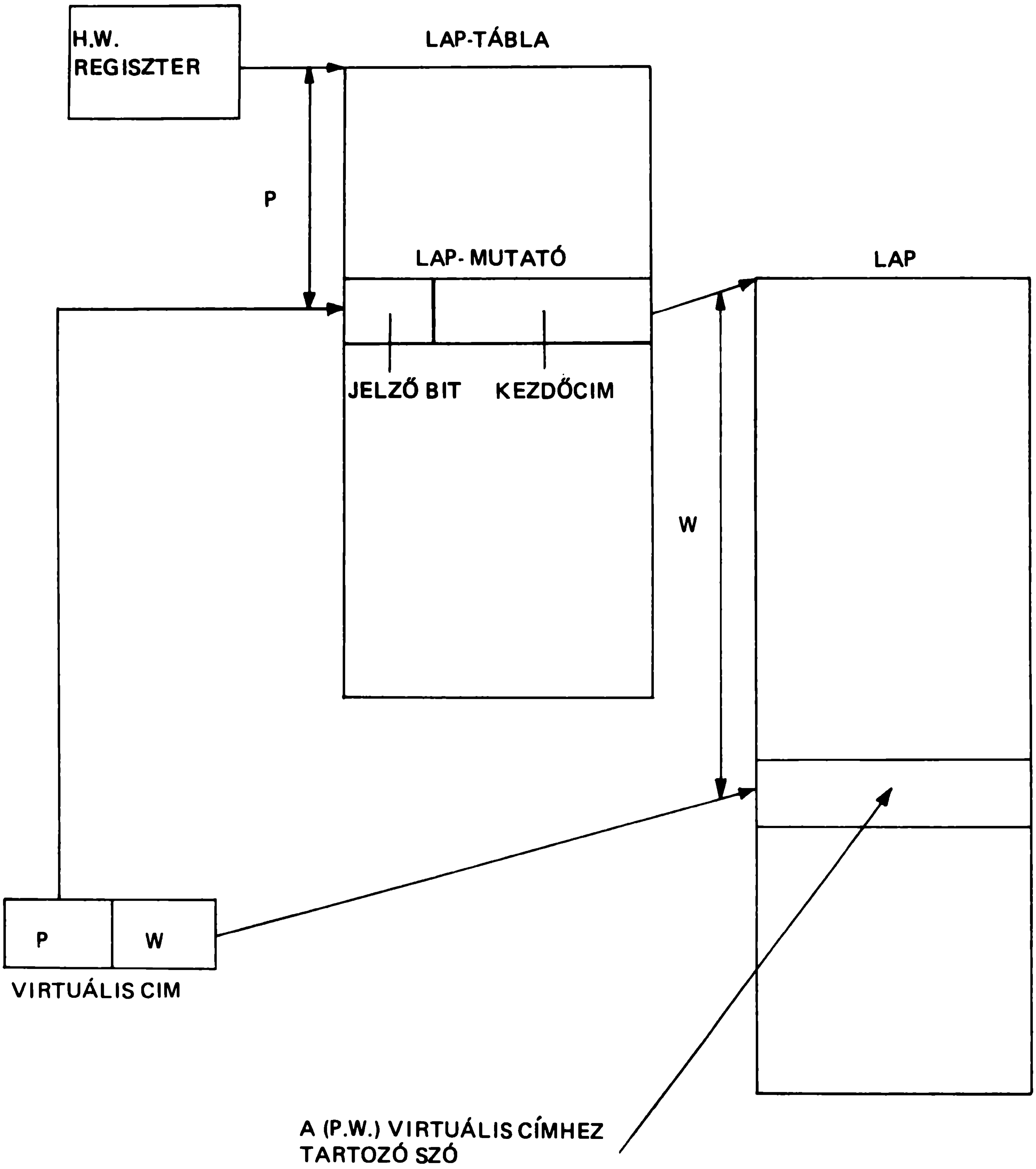
Az asszociativ tárat felhasználó cimkonverzió folyamatát a 11. ábra szemlélteti.

### *3.7.1. Lapok megosztott használata*

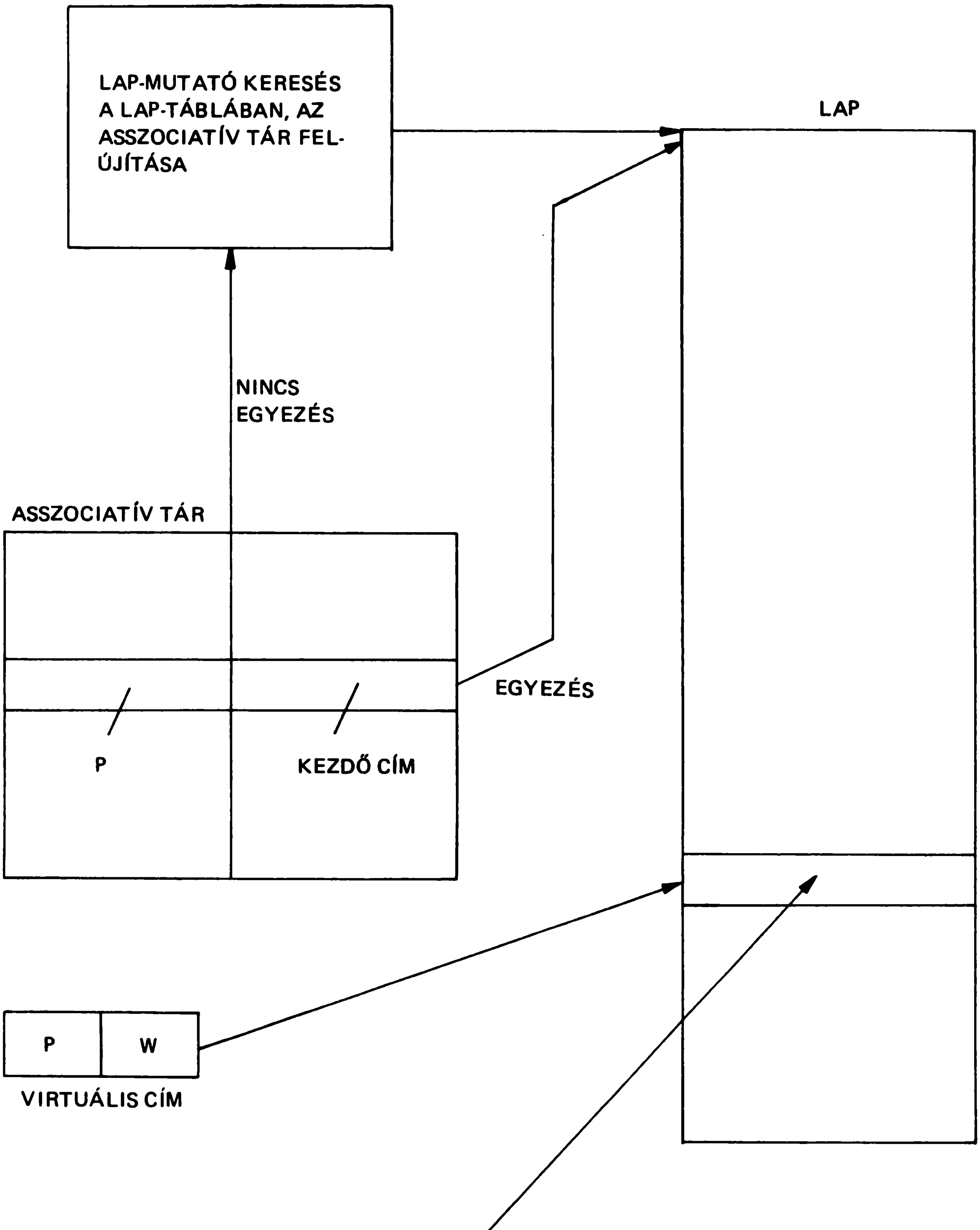
A lapkezelés alapvető jellemzője - az, hogy mindegyik program több, nem folytatólagos darabot foglal el az operativ memóriában - felhasználható arra, hogy két vagy több program közös lapokat használjon (ld. 12. ábra).

A közösen használt lapokon például könyvtári rutinok, vagy adatok helyezkednek el, melyek több példányban való tárolása feleslegessé válik.

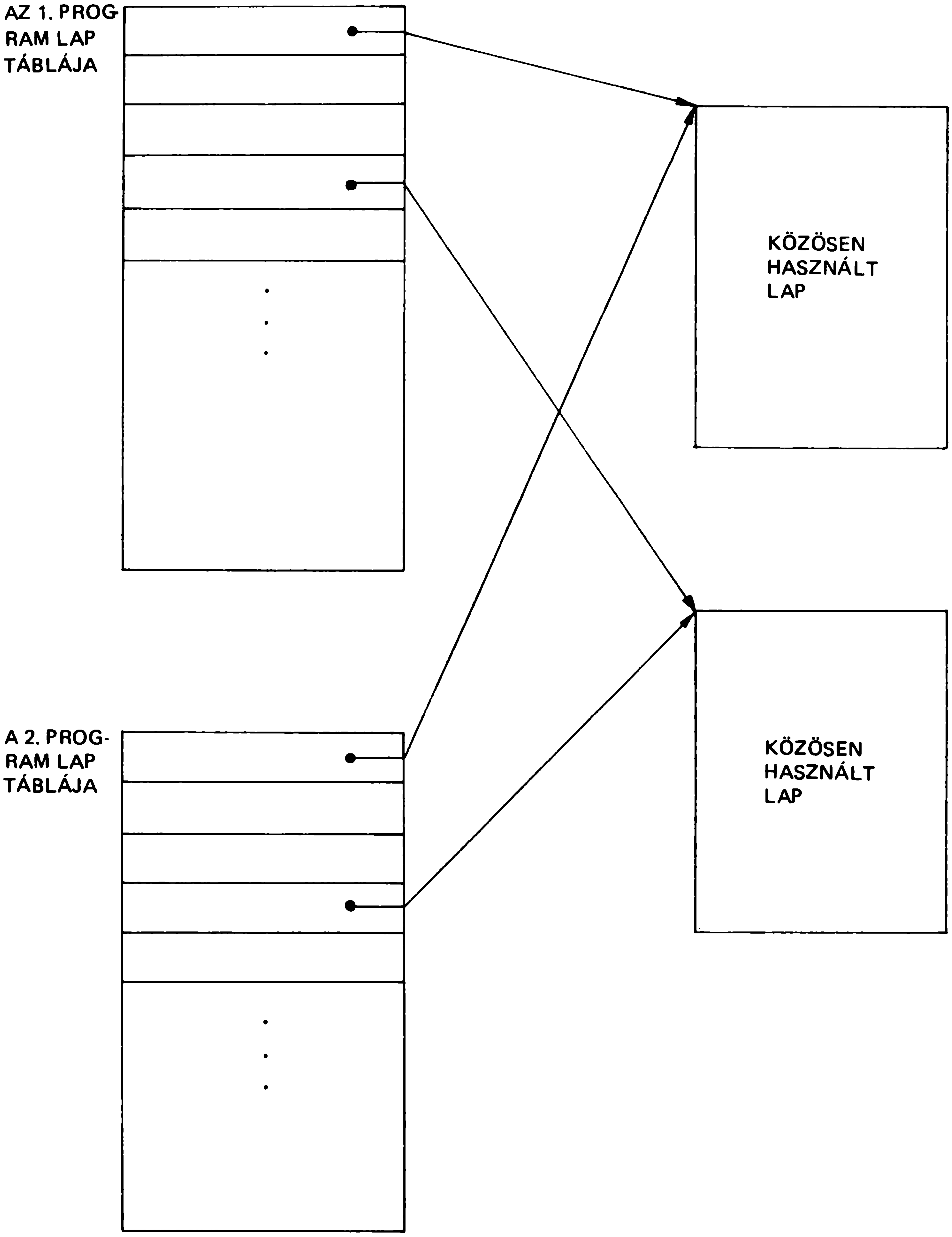
A megosztott laphasználatból adódó elsődleges probléma, hogy a közösen használt lapoknak a háttértárból való behozatalakor, vagy


H.W.
REGISZTER
LAP-TÁBLA
P
LAP-MUTATÓ
LAP
JELZŐ BIT
KEZDŐCIM
W
P
W
VIRTUÁLIS CIM
A (P.W.) VIRTUÁLIS CÍMHEZ
TARTOZÓ SZÓ

*10. ábra*
*Cimkonverzió*


LAP-MUTATÓ KERESÉS A LAP-TÁBLÁBAN, AZ ASSZOCIATÍV TÁR FELÚJÍTÁSA
LAP
NINCS EGYEZÉS
ASSZOCIATÍV TÁR
EGYEZÉS
P
KEZDŐ CÍM
P
W
VIRTUÁLIS CÍM
A (P.W.) VIRTUÁLIS CÍMHEZ TARTOZÓ SZÓ

*11. ábra*
*Cimkonverzió asszociativ tárral*


AZ 1. PROGRAM LAP TÁBLÁJA
KÖZÖSEN HASZNÁLT LAP
A 2. PROGRAM LAP TÁBLÁJA
KÖZÖSEN HASZNÁLT LAP

*12. ábra*
*Közös lapok*

oda való kiirásakor a rájuk mutató lap-tábla elemeket minden laptáblában módositani kell. Ennek megoldására több módszer létezik:

a) A legegyszerübb módszer, hogy minden lapmozgatáskor az összes lap-táblát végignézzük. Ez különösen akkor lassu, ha egy lapról nem lehet előre eldönteni, hogy több program használja-e, vagy nem. Valamit lehet javitani a módszeren, a közösen használt lapok lap-mutatóinak láncolásával, igy egy lap mozgatásakor az öszszes, esetleges lap-mutatók viszonylag egyszerüen megtalálhatók.

b) Egy másik módszer szerint a közösen használt lapok számára egy külön lap-tábla kerül kitöltésre, a lap-mutató ebből a táblából választódik ki, ha a lap-cim egy rögzitett intervallumban van. Ebben az esetben a közösen használható lapok száma korlátozott, azonban ez általában nem nagy hátrány, mivel gyakorlatilag elegendő csupán a leggyakrabban használt könyvtári rutinok megosztott használata.

c) A harmadik módszer minden laphoz egy fő-lap-mutató bevezetését jelenti. A normál lap-mutatóban lévő jelző egyik állapota a közösen használt lapok esetén azt jelenti, hogy a kezdőcim a fő-lap-mutató cime. Ilymódon a közösen használt lapokra való hivatkozáskor ugyan közbe kell iktatni a fő-lap-mutatót, viszont a lap mozgatásakor csak a fő-lap-mutatót kell módositani.

A másik fontos feltétel, amit programok vagy adatok megosztott használatakor teljesiteni kell, az, hogy ezen a lapok tartalmát egyik program se változtathassa meg, sem szándékosan, sem véletlenül. Ezt a problémát a lap-mutatónak egy un. "hozzáférés szabályzó" bittel való kiegészitése árán lehet megoldani. Ez a bit a közösen használt lapok tartalmának megváltoztatását hardware (vagy software) uton letiltja.

### *3.7.2. Virtuális tár*

A lapkezelés bevezetésében (ld. 3.7. fejezet) felsorolt - az egyéb memória kezelő módszerek által megoldatlanul hagyott - problémák közül a harmadik, nagy programok felhasználó által szervezett overlay technika nélküli kezelése a virtuális tár elvének bevezetésével oldható meg.

A programozó folyamatosan növekvő cimeket rendel az utasitásokhoz, nem törődve azzal, hogy közben átlépi a gép cimtartományának felső határát. Ilymódon a felhasználó a programjának irásakor elsősorban nem a tényleges operativ memória méretét veszi figyelembe, hanem egy ennek többszörösét kitevő virtuális tárat használ. Természetesen, a tényleges memória méretet is figyelembe kell venni, annyiban, hogy bármely adott pillanatban a program memóriaigénye kielégithető legyen, különben az állandó információcsere a háttértárral elviselhetetlenül lelassitja a futást.

A virtuális memóriát akkor is jól használhatjuk, ha soha, vagy csak nagyon ritkán lépjük tul a tényleges memória határát. Például forditóprogramok esetében célszerü az előre nem ismert változó nagyságu táblázatok számára minél több helyet lefoglalni, nehogy forditás közben beteljenek. A táblázatokhoz rendelt lapok jó része talán soha nem kerül felhasználásra és nem is foglal el memóriát, csupán nagymértékben kiterjesztheti a forditóprogram alkalmazhatóságát, feldolgozási kapacitását.

A virtuális tár kezelésének technikája a következő:

Bevezetjük a több lapból álló memória terület, a szegmens fogalmát.

A virtuális cimek három részre oszlanak: szegmens-cim, lapcim, szó-cim. A lap-táblák fölé egy szegmenstábla kerül, amely szegmensenként egy jelzőbitet és a megfelelő laptábla kezdőcimét tartalmazza. A jelzőbit szerepe azonos a laptábla elemeiben, a lapmutatókban használtéval.

A cimkonverzió menetét a 13. ábra mutatja be a korábban ismertetett asszociativ tár használatát feltételezve.

Mivel a cimkonverzió bonyolultabbá vált, a táblákban való keresés gyorsitására az asszociativ tár vagy más hasonló célu módszer jóformán nem nélkülözhető.

A virtuális tárkezelés lehetőségei a jelenleg elterjedt operációs rendszerekben még távolról sincsenek kihasználva. A gyors fejlődés ellenére sem láthatók be teljesen például a Dijkstra által alkalmazott stratégia következményei, melyben a programozónak néhány - a supervisor által kezelt - lap-tábla segitségével a lap-mutatókon való manipulációs lehetőséget adnak. /Irod. 7./

A lapkezelés általános stratégiáinak áttekintése után bemutatjuk ezen technika megvalósitásában alkalmazott fontosabb algoritmusokat.

### *3.7.3. Lapkezeléssel kapcsolatos algoritmusok*

Annak az alapelvnek az elfogadása, hogy a futó programok számára szükséges lapokat nem előre meghatározott, hanem tetszőleges helyre vihetjük be az operativ memóriába, természetes következményként hozta magával a laphelyettesitési algoritmusokat. Ezen algoritmusok célja, hogy eldöntsék, hogy az ujonnan behozott lap melyik benlevőt irja felül, melyik lap legyen az "áldozat".

Az elméletileg tökéletes helyettesitési elv az, hogy az áldozat olyan lap legyen, amelyre a leghosszabb ideig egyik program sem fog hivatkozni. Ezt gyakorlatilag előre nem lehet meghatározni, tehát valamilyen közelitő megoldást kell keresni.

A helyettesitési elv ilymódon átfogalmazott formája egy olyan lap áldozatként való kijelölését tüzi ki célul, amelyre várhatóan egyik program sem fog hivatkozni valamilyen, számottevő ideig.

A választás a programok multbeli és jelenlegi viselkedéséből levont következtetések alapján történik. Abból a célból, hogy a programok előző viselkedéséről információt kapjunk, egy ugynevezett "használati bit" bevezetésére van szükség. A használati bit a lapmutatóban helyezkedik el, és a hardware 1-re állitja értékét, ha valamelyik program a lapra hivatkozik. A használati bit értékét a supervisor szabályos időközönként vizsgálja és törli. Ilyen módon megtudható például, hogy egy lap mennyi ideje nem került felhasználásra.

A használati biten kivül szokás egy "irás" bitet is bevezetni, amely szintén a lap-mutatókban helyezkedik el, és akkor veszi fel az 1 értéket, ha valamelyik program a lap tartalmát megváltoztatja. Az irásbitet a lap behozatalakor a supervisor nullára állitja. Ha a lap helyettesitésekor az irásbit értéke még mindig nulla, egy átvitel megtakaritható, mivel az áldozatot nem kell visszamásolni a háttértárba.

Az áldozat kiválasztására sok algoritmus létezik, melyek közül a gyakran használtak a következők:

a) A legrégebben használt lap helyettesitése azon a feltételezésen alapszik, hogy ha egy lapra már hosszu idő óta nem történt hivatkozás, akkor valószinü, hogy még számottevő ideig nem is fog történni. A használati bit segitségével egy *t* számláló alakitható ki, mely azt mutatja, hogy a lapot mennyi ideje nem használták (*t* értékét a használati bit periodikus vizsgálatának eredményei alapján lehet meghatározni).

Áldozatként a legnagyobb *t* értékkel rendelkező lap kerül kiválasztásra.

b) Talán a legegyszerübb helyettesitési algoritmus az, amelyben mindig a legrégebben az operativ memóriában tartózkodó lap az áldozat (FIFO, azaz First-In-First-Out technika).

Az algoritmus elméleti alapját az a következtetés képezi, hogy a régen behozott lapok valószinüleg kevésbé vesznek részt az "eseményekben", mint az éppen most behozottak.

Az algoritmus népszerüségét azonban nem annyira az elméleti megalapozottsága, mint megvalósitásának egyszerüsége eredményezte. Mindössze egy - az áldozatra mutató - regiszterre van szükség. Az áldozat kicserélésekor a regisztert egyszerüen a következő lapra kell átállitani, illetve az utolsó lap után az elsőre.

c) A legrégebben használt lap helyettesitése és a FIFO elv kombinációjából nyerhető egy könnyen megvalósitható, jól használható algoritmus:

A FIFO-nál emlitett ciklikus mutatót alkalmazzuk, de az áldozat kiválasztásánál megvizsgáljuk a használati bitet is. Ha ez egy, törlődik és a mutató továbblép. Ha nulla, ez azt jelenti, hogy a lapra az előző vizsgálat óta egyik program sem hivatkozott, tehát ő lesz az áldozat.

Az algoritmus megvalósitása még a hardware kezelte használati bit hiányában is egyszerü és gazdaságos. Ilyen esetekben a lap-mutatóban lévő jelző bit (lásd 10. ábra) értékét szokták felhasználni olymódon, hogy ha a mutató a lapra mutat, a jelző "nincs bent" értékre állitódik, de az operativ memóriában marad. Amikor a mutató legközelebb ismét a lapra mutat és a jelző értéke nem változott (ami azt jelenti, hogy időközben nem hivatkozott egyetlen program sem a lapra), a lap lesz az áldozat.

Az áldozat kiválasztása mellett a rendszer által megoldandó másik probléma a tétlenségi idő csökkentése. A tétlenségi idő a futó program uj lap behozására való várakozási ideje, amely alatt a program nem csinál semmit, de leköti az erőforrásokat. Ide vonatkozó számitások szerint a tétlenségi idő csökkentése nagyon fontos, mert ha ez a futási idő egy bizonyos (sajnos nem is magas) hányadát meghaladja, a program tárfelhasználása nagyobb lesz, mintha a teljes program a tárban lett volna futás alatt, ez pedig cáfolná a lap technika által igért előnyöket, feleslegessé tenné alkalmazását. (Egy program tárfelhasználásán a lefoglalt tár nagyságának és az ott eltöltött időnek a szorzatát értjük.)

A tétlenségi idő csökkentésére ajánlott és használt algoritmusok közül az alábbiak érdemelnek emlitést:

a) A legegyszerübb módszer a tétlenségi idő csökkentésére a következő áldozat előre való kiválasztása. Ha az áldozatot vissza kell másolni a háttértárba, ez a módszer jó esélyt biztosithat arra, hogy mikor az uj lapra ténylegesen szükség van, a kimásolás már befejeződött, és igy a tétlenségi idő csak az uj lap behozatali idejére korlátozódik.

b) Egy másik módszer azon a felismerésen alapszik, hogy a viszonylag hosszu időre - például a felhasználóval való interaktiv üzenetváltás miatt - felfüggesztett programoknál a program folytatásakor célszerü a teljes felfüggesztés előtti állapotot egy lépésben visszaállitani, azaz visszahozni a felfüggesztés előtt használt összes lapot, mivel ez rövidebb tétlenségi időt eredményez, mint a lapoknak egyenként - a konkrét igény alapján való - visszahozatalkor fellépő tétlenségi idők összege.

c) A lapok láncolásának módszere a felhasználó számára biztosit lehetőséget arra, hogy segitséget nyujtson az operációs rendszernek a tétlenségi idő csökkentésében. A módszer azon a felismerésen alapszik, hogy a programok olyan szegmensekre oszlanak, hogy egy szegmens valamely cimére való hivatkozást rövidesen a szegmens más cimeinek felhasználása követi. (Ilyen szegmensek például egy-egy szubrutint, táblázatot, adattömböt tartalmaznak.) A lapok szemszögéből nézve arról van szó, hogy ha be kellett hozni egy szegmens egyik lapját a háttértárból, rövidesen be kell majd hozni a többi lapot is, amelyik a szegmenshez tartozik, tehát célszerü ezt az átvitelt egyszerre elvégezni az elsőnek fellépő igény kielégitésekor.

A módszer megvalósitása a lapmutatókban egy láncolási bit elhelyezését igényli. A láncolási bit értéke 1, ha a lap teljes egészében egy szegmensben van, nulla, ha szegmens határt tartalmaz. Ha egy olyan lapot kell behozni, melynek láncolási bitjén 1 áll, vele együtt a szegmens többi teljes lapja is behozatalra kerül. Nullás értékü láncolási bittel rendelkező lapok cseréjének esetén másik lap nem jön be, mivel nem lehet tudni, hogy a két szegmens közül melyikre van szükség. A módszer különösen kis lapmérettel dolgozó rendszerekben hatásos.

Az előzőekben bemutatott algoritmusok tanusága szerint komoly erőfeszitést igényel az operációs rendszer jó hatásfokkal való müködésének biztositása. A gyakorlat azt mutatja, hogy a rendszer hatásfoka nagy mértékben függ a terheléstől, azaz a feldolgozás alatt álló job-ok mennyiségétől és minőségétől. A legtöbb rendszernél minden igyekezet ellenére előállhatnak olyan tulterhelési helyzetek, amikor a hasznos munkavégzés lecsökken, a tétlenségi idő megnövekszik. A tulterhelés teljes kiküszöbölése elvileg nem valósitható meg a felhasználók korlátozása nélkül, azonban léteznek módszerek, melyek alapján az operációs rendszer számottevő mértékben "megvédheti" magát a tulterheltségtől.

Az algoritmusok közül csak egyet emlitünk, amely jól kifejezi a probléma lényegét, a lapcsere korlátozást.

A módszer lényege az, hogy az adott hardware, software és felhasználói környezet figyelembevételével, a rendszer müködéséből vett gyakorlati adatok (pl. lapcsere elvégzésére forditott teljes idő, ebből i/o csatorna idő, központi egység idő stb.) alapján küszöbértékek határozhatók meg az időegységre eső lapcserék számára vonatkozóan. Ha a lapcsere intenzitás a felső küszöbértéket meghaladja, a rendszer terhelését egy program felfüggesztésével enyhiteni kell. Ha a lapcsere intenzitás az alsó küszöbérték alá esik, a terhelés növelhető, például egy előzőleg felfüggesztett program folytatásával.

### *3.7.4. Változó nagyságu lapok*

A lapkezelés és ezzel a memória gazdálkodási stratégiák kérdésének lezárásaként egy olyan problémát tárgyalunk, amely alapvető jelentőségü, de megoldása csak az eddig tárgyalt módszerek alapján látható világosan. A megfelelő lap méret kiválasztásáról van szó, amely nagymértékben befolyásolja az egész rendszer teljesitményét. Ha kis lapokkal dolgozunk, a lap-táblák nagyok lesznek és a sok lapot elfoglaló programok háttértárból való betöltése, illetve visszavitele sok input-output müveletet igényel. (Ezen a problémán igyekszik segiteni a lapok láncolásának módszere.)

Ha a lapméret nagy, lesznek olyan programok, melyek nem töltenek ki egy lapot, vagy olyan folyamatos területet igényelnek, amely laphatárt is tartalmaz. Ez azt jelenti, hogy a lapok egyik, másik, vagy mindkét végén számottevő ideig kihasználatlan memória területek maradnak fenn, amit a lapkezelő rendszer nem tud hasznositani a lapméret szabta merev határok következtében.

A gyakorlatban egy szótól 8192 szóig jóformán minden lapméretet javasoltak, vannak olyan rendszerek is, melyek két vagy több méret közül kinálnak választást.

A probléma általános megoldását kézenfekvő változó nagyságu lapok bevezetésével megközeliteni olymódon, hogy a lapok mérete a programozó által meghatározott logikai program szegmensek méretéhez igazodik. Ez a technika sikeres alkalmazásra talált a Burroughs B 5500 rendszerben /Irod. 8. / Minden lap egy főprogramnak, egy szubrutinnak, vagy egy adatvektornak felel meg.

Várható módon a változó nagyságu lapok kezelése az előzőekben bemutatott algoritmusok változatlan alkalmazását nem engedi meg, az ebből fakadó néhány fontosabb probléma röviden összefoglalva a következő:

a) Lapok elhelyezése az operativ memóriában.

Rögzitett lapméret esetén ez a kérdés fel sem merült, a háttértárból bejövő lapot akármelyik üres helyre el lehetett helyezni. Két alapvető stratégia létezik, az egyik a bejövő lapot a neki méret szerint legjobban megfelelő területen helyezi el (Best Fit), a másik az elsőként adódó elég nagy helyen (First Fit). Kisérleti eredmények alapján általában az utóbbi stratégia biztosit nagyobb teljesitményt a rendszer számára. /Irod. 9. /

b) Háttértár gazdálkodás

A változó méretü lapok háttértárban tárolása azért okoz problémát, mert a tárak rögzitett szektor szerkezete következtében igen rossz hatásfoku tárhasználat következhet be, hiszen az irás és olvasás csak szektorhatáron kezdődhet.

c) Szegmentált lapok

A fentiekben emlitett kérdések megoldását elősegitő módszer, melynek lényege az, hogy a változó nagyságu lapokat kis, fix méretü szegmensekre osztják. A háttértárral való információ cseréjénél a lapok mozognak, az operativ tárban azonban a fix szegmensek külön-külön (nem okvetlenül folytatólagos helyen) kerülnek elhelyezésre. A technikai megoldás lényegében a virtuális tárkezelésével (lásd 3. 7. 2. ) azonos.

Meglehetősen bonyolult hardware-t és software-t igényel, például a lap-táblákon kivül szegmens táblák kezelésére is szükség van.

# 4.

# A gépi architektura és az operációs rendszerek kölcsönhatása

A különböző tárkezelési módszerek áttekintése során láttuk, hogy a fejlettebb, a felhasználók számára több lehetőséget nyujtó stratégiák ujabb és ujabb hardware eszközöket igényelnek. Ezzel kapcsolatban felmerülő első kérdés, hogy valóban nélkülözhetetlenek-e ezek a hardware megoldások, illetve milyen hatást gyakorolnak az architektura adottságai az operációs rendszerekre.

Az architektura jellemzői közül válasszuk ki azt a néhányat, amelyek az operációs rendszerre, illetve a rendszer által a felhasználó számára megteremtett virtuális gépre a legnagyobb hatással vannak.

## *4.1. A központi egység hatása*

A központi egység tulajdonságai közül az utasitásrendszer emelhető ki, mint a virtuális gépre legnagyobb hatással lévő hardware jellemző.

Az operációs rendszerek többségének esetében a virtuális központi egység az eredeti hardware központi egység egy részhalmaza, melyben bizonyos privilegizált funkciók rejtve vannak a felhasználók elől.

Tulajdonképpen elméletileg semmi akadálya annak, hogy a virtuális gép utasitásrendszere teljesen más legyen, mint az eredeti utasitásrendszer. Ilyen rendszerek müködnek is kutató tevékenységet folytató intézményeknél. A megoldás lényege az, hogy a virtuális gép utasitásrendszerét egy értelmező program hajtja végre az eredeti gépen. Kellő szintü értelmező esetén a felhasználónak sejtelme sincs az eredeti utasitásrendszerről, szóhosszáról, regiszterekről stb.

Ez a módszer azonban a gyakorlati számitástechnikában nem alkalmazható, mert tul költséges. A virtuális gép nagyon lassu lesz az eredetihez képest, ami legfeljebb csak kutatási alkalmazásoknál viselhető el.

Ennek köszönhető, hogy az eredeti központi egység sok tulajdonsága általában megtalálható a virtuális gépnél is.

Mivel a gépek utasitásrendszerét általános célura tervezik, az eredeti gép utasitásainak jó része jól felhasználható a virtuális gépnél. Példaképpen a géptől független nyelvek (pl. FORTRAN, ALGOL) implementálását emlitjük. Ezek a nyelvek - igaz, hogy különböző hatékonysággal - szinte minden gépen implementálhatók.

A központi egység tulajdonságai tehát gyakorlati - gazdaságossági - szempontok alapján számottevő mértékben érvényesülnek a virtuális gépen.

Az architektura más sajátosságai, mint pl. a 3. fejezetben tárgyalt memóriakezelési módszerekhez kapcsolódó hardware szolgáltatások talán még erősebb hatással vannak az operációs rendszerre, illetve a virtuális gépre.

## *4.2. A memóriakezelés hatása*

Az architektura által biztositott memóriakezelésnek a virtuális gép tulajdonságaira gyakorolt hatásának vizsgálat több szempontból fontos.

Egyrészt segitséget nyujthat létező rendszerek gyenge pontjainak felderitésében a helyes fejlesztési szempontok meghatározása céljából, másrészt nyilvánvalóvá teszi, hogy milyen árat kell fizetnünk azért, hogyha operációs rendszer segitségével az architekturát valamilyen nagyon eltérő, virtuális géppé akarjuk átalakitani.

Ennek alátámasztására vizsgáljuk meg az architektura által biztositott, különböző szintü tárkezelési módszereket a felhasználó által támasztott igények szempontjából:

a) Közvetlen cimzés

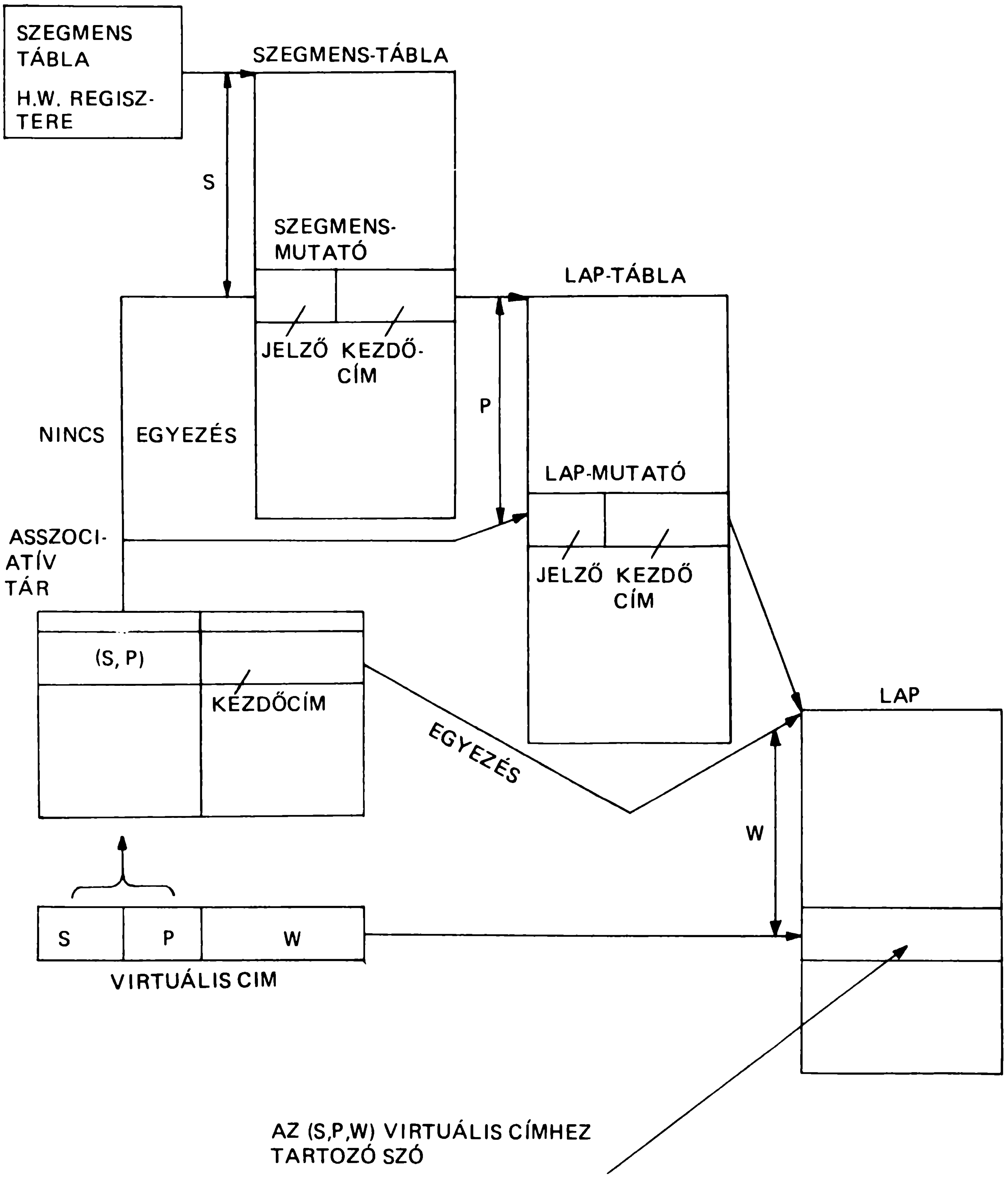
Ez a 3.1. fejezetben ismertetett stratégia, mint láttuk, jól beválik batch üzemmódu alkalmazásoknál. Ha azonban egy több felhasználós, multiprogramozásu környezetet akarunk kiépiteni erre az architekturára, a következő problémával találjuk magunkat szembe:

mivel a programoknak a memóriához való hozzáférése nincs korlátozva, minden felhasználó a teljes operativ tárat birtokolja, tehát program váltáskor az egész memóriát ki kell menteni, illetve be kell hozni a háttértárból.

Az ebből származó időtöbblet megbecsülhető, és a rendszer munkaidejének 50%-át is kiteheti. Ez olyan veszteség, amit csak kisérleti körülmények között üzemelő rendszereknél engedhetünk meg. A felhasználót nehezen lehetne rábeszélni, hogy egy korszerübb üzemmód érdekében áldozza fel a számitógép potenciális teljesitményének felét.

b) Multiprogramozás és áthelyezhetőség

A 3.2., illetve 3.3. fejezetben bemutatott tárkezelési stratégiák implementálása, mint láttuk, közvetlen cimzés esetén nagyon gazdaságtalan.


SZEGMENS TÁBLA H.W. REGISZTERE
SZEGMENS-TÁBLA
S
SZEGMENS-MUTATÓ
JELZŐ KEZDŐCÍM
LAP-TÁBLA
P
NINCS
EGYEZÉS
LAP-MUTATÓ
ASSZOCIATÍV TÁR
JELZŐ KEZDŐ CÍM
(S, P)
KEZDŐCÍM
EGYEZÉS
LAP
W
S
P
W
VIRTUÁLIS CIM
AZ (S,P,W) VIRTUÁLIS CÍMHEZ TARTOZÓ SZÓ

*13. ábra*
*Cimkonverzió szegmentálás esetén*

Vizsgáljuk meg, hogy milyen tulajdonságokkal kell felruházni az architekturát, hogy ezek a stratégiák gazdaságosan realizálhatók legyenek.

Az első követelmény: az operativ memóriát több felhasználó megosztva birtokolja, olymódon, hogy ameddig az egyik felhasználó programja fut a többiek programjai a memóriában maradhassanak. E probléma hardware megoldása viszonylag egyszerü (pl. a széles körban használt memória-kulcsok segitségével). Az operativ memória alkalmasan választott nagyságu blokkjaihoz kulcsok tartoznak, melyek egyértelmüen jellemzőek a blokkokat elfoglaló programra.

A virtuális gép közvetlen cimeket generál. A kulcsok segitségével egy megszakitás generálható, ha ez a program nem a saját kulcsával rendelkező cimhez fordul.

Az operativ memória megosztott használatához szorosan kapcsolódik a második követelmény, amit az architekturának ki kell elégitenie: a programok áthelyezhetősége. Ezzel a lehetőséggel ugyanis nagymértékben növekszik a rendszer rugalmassága és teljesitőképessége, mivel igy a központi memória különböző időpontokban különböző programok által lehet elfoglalva és egy felhasználói program futása nincs olyan feltételhez kötve, hogy a memória egy bizonyos darabja szabad legyen.

A programok áthelyezhetővé tételének egyik lehetséges, de nem könnyen járható utja pusztán software eszközökkel is megvalósitható: a program áthelyezésekor egy supervisor rutin módositja a megfelelő utasitások cimeit a kivánt mértékben. A fő problémát az okozza, hogy meg kell különböztetni az utasitásokat a program konstansaitól, az adatoktól, ezeket ugyanis nem szabad a program áthelyezésekor megváltoztatni. Ezen tulmenően létezhetnek olyan utasitások is, amelyek hatása független tárolási cimüktől, tehát szintén változatlanoknak kell maradniuk. Az ilyen elven müködő rendszerek a programok utasitásainak és az adatoknak szigoru szétválasztásán alapulnak. Az architektura által nyujtott lehetőségek felhasználásával megvalósitott programáthelyezés - a virtuális cimzés egyik egyszerü változata - minden szempontból jobb megoldás.

A program csak virtuális cimeket generál és ezekhez az utasitás végrehajtása előtt hozzáadódik egy bázis regiszter tartalma. A virtuális cimek tulajdonképpen relativ cimek (a program elejéhez képest). A kapott tényleges cim összehasonlitható egy másik - a programhoz tartozó cimtartomány felső határát tartalmazó - regiszter tartalmával, a memóriának a programhoz nem rendelt része védelme érdekében (ld. a 3.2. fejezetben bevezetett LA és LF regisztereket).

### c) Általános virtuális cimzés

A 3.7. fejezetben ismertetett stratégiának, a lapkezelés különböző változatainak megvalósitásához, az ott leirtakon kivül tulajdonképpen nincs szüksége további hardware támogatásra. A lap-táblákban való kereséskor alkalmazható asszociativ memória ugyan hasznos, de korántsem nélkülözhetetlen eszköz.

Ebből az a következtetés vonható le, hogy az architekturának az operációs rendszerekre gyakorolt hatása ciklikusan érvényesül. A gyakorlati felhasználás által támasztott igények már viszonylag alacsony szinten is olyan problémákhoz vezetnek, melyeknek pusztán software eszközökkel való megoldása nehézkes, lassu, nem gazdaságos (pl. megosztott tárhasználat, program áthelyezhetőség). E problémákra tehát az architekturának kell jó megoldást adnia. Amikor viszont megszületik a megfelelő hardware szolgáltatás, az erre épülő software további lépéseket tesz előre és az ujabb rendszerek egy darabig architekturális támogatás nélkül is gazdaságosan fejleszthetők, amint ezt a lapkezelés példája mutatja. A fejlődés valamilyen szintjén azután megint olyan problémák merülnek fel, amelyek leküzdése már csak ujabb hardware eszközök bevezetésével oldható meg a gyakorlati követelményeket kielégitő módon.

# Irodalomjegyzék[1]

/ 1/ Denning, P.J.: The Developing Theory of Operating Systems. Operating Systems. International Computer State of the Arts Report. INFOTECH 1972. Maidenhead, England.

/ 2/ Dijkstra, E.W.: Hierarchial Ordering of Sequential Processes. Operating Systems Techniques. Academic Press, 1972. London.

/ 3/ Nevell, G.B.: GEORGE 3, the Compleat Operating System. Operating Systems. International Computer State of the Arts Report. INFOTECH 1972. Maidenhead, England.

/ 4/ Strachey, C.: The Interaction of Software Engineering and Machine Structure. The fourth Generation. International Computer State of the Arts Report. INFOTECH 1972. Maidenhead, England.

/ 5/ Weegenaar, H.J.: Job Control Languages: What the User Really Requires. Operating Systems, International Computer State of the Arts Report. INFOTECH 1972. Maidenhead, England.

/ 6/ Hoare, C.A.R.: A Survey of Store Management Techniques. Operating Systems Techniques. Academic Press 1972. London.

---

[1] Az irodalomjegyzékben a következő rövidítések szerepelnek:

ACM - Association for Computing Machinery
IEEE - Institute for Electrical and Electronics Engineers
IEEETC - IEEE Transactions on Computers
CACM - Communications of the ACM
JACM - Journal of the ACM
CS - Computing Surveys (ACM)
FJCC - Fall Joint Computer Conference
SJCC - Spring Joint Computer Conference
2SOSP - Second Symposium on Operating Systems Principles

/ 7/ Dijkstra, E.W.: The structure of the T.H.E. Multiprogramming System. Communications of the ACM, May, 1968.

/ 8/ Burroughs: A Narrative Description of the B 5500 Disk File Master Control Program. Burroughs Corp. Detroit, 1966.

/ 9/ Kunth, D.E.: The Art of Computer Programming Addison Wesley, 1969. Reading, Massachusetts.

/10/ Abate, J., and Dubner, H.: Optimizing the Performance of a Drum-Like Storage. IEEE Trans. C-18, 11 (Nov. 1969), 992-997.

/11/ Abrams, M.: A Comparative Sampling of the Systems for Producing Computer-Drawn Flowcharts. Proc. 1968. ACM Nat. Conference.

/12/ Arden, B. and Boettner, D.: Measurement and Performance of a Multiprogrammed System. Proc. 2SOSP Oct. 1969.

/13/ Belady, L.A.: A Study of Replacement Algoritms for Virtual Storage Computers. IBM Sys. J. 5, 2, 1966, 78-101.

/14/ Bell, C.G. and Newell, A.: Computer Structures: Readings and Examples. Chapt. 3, McGraw Hill, 1971. New York.

/15/ Bensoussan, A., Clingen, C.T., and Daley, R.C.: The Multics Virtual Memory. Proc. 2SOSP, Oct. 1969.

/16/ Bernstein, A.J. et al.: Process Control and Communication. Proc. 2SOSP, Oct, 1969.

/17/ Bétourné, C., et al.: Process Management and Resource Sharing in the Multiaccess System "ESOPE". CACM 13, 12, Dec. 1970.

/18/ Calingaert, P.: System Performance Evaluation Survey and Appraisal. CACM Jan. 1967.

/19/ Clark, W.A.: The Functional Structure of OS/360, Part III. Data Management IBM Sys. J. 5, 1, 1966, 30-51.

/20/ Coffman, E.G., Elphick, M., Shoshani, A.: System Deadlocks. CS 3, 2, June 1971.

/21/ Coffman, E., Kleinrock, L.: Computer Scheduling Measures and Their Countermeasures. AFIPS Conf. Proc. 32, 1968 SJCC.

/22/ Corbato, F.J., Vyssotsky, V.A.: Introduction and Over-View of the Multics System. AFIPS Conf. Proc. 27. 1965 FJCC, 185-196.

/23/ Crisman, P.A.: The Compatible Time-sharing System; A programmer's Guide. Mit Press, 2nd Edition, Cambridge, Mass. 1965.

/24/ Daley, R.C., Dennis, J.B.: Virtual Memory, Processes, and Sharing in MULTICS. CACM 11, 5 May 1968. 306-312.

/25/ Daley, R.C., Neumann, P.G.: A General-Purpose File System for Secondary Storage. AFIPS Conf. Proc. 27. 1965 FJCC, 213-229.

/26/ Denning, P.J.: Effects of Scheduling on File Memory Operations. AFIPS Conf. Proc. 30. 1967 SJCC, 9-22.

/27/ Denning, P.J.: The Working Set Model for Program Behaviour, Comm. ACM 11, 5, May 1968, 323-333.

/28/ Denning, P.J., Thrashing: Its Causes and Prevention AFIPS Conf. Proc. 33, 1968 FJCC, 915-922.

/29/ Denning, P.J.: Equipment Configuration in Balanced Computer Systems. IEEETC C18, Nov. 1969. 1008-1012.

/30/ Denning, P.J.: Virtual Memory. CS 2, 3, Sept. 1970, 153-189.

/31/ Dennis, J.B.: Segmentation and the Design of Multiprogrammed Computer Systems. JACM 12, 4, Oct. 1965, 589-602.

/32/ Dennis, J.B.: A Position Paper on Computing and Communications. CACM 11, 5, May 1968. 370-377.

/33/ Dennis, J.B., Van Horn, E.C.: Programming Semantics for Multiprogrammed Computations. CACM, 9, 3, March, 1966, 143-155.

/34/ Dennis, J.B. et al.: Computation Structures. Chapt. 8. Class notes for Subject 6. 232, MIT, 1970. (Available from MIT, Rm 4-213, Cambridge, Mass. 02139).

/35/ Dijkstra, E.W.: Solution of a Problem in Concurrent Programming Control. CACM 8, 9, Sept. 1965, 569.

/36/ Dijkstra, E.W.: Cooperating Sequential Processes. Programming Languages (F. Genuys, ed.), Academic Press (1968), 43-112.

/37/ Dijkstra, E.: A constructive approach to the problem of program correctness. BIT 8, 1968.

/38/ Fano, R.M., Corbato, F.J.: Time Sharing on Computers Sci. Amer. 215, 3, Sept. 1966, 129-140.

/39/ Foster, J.M.: List processing. MacDonald and Co., London, 1967.

/40/ Fuchs, E., Jackson, P.E.: Estimates of Random Variables for Certain Computer Communications Traffic Models. CACM 13, 12, Dec. 1970.

/41/ Gear, C.W.: Computer Organization and Programming. McGraw Hill, 1969. New York.

/42/ Graham, R.M.: Protection in an information Processing Utility. CACM 11, 5, May 1968, 368.

/43/ Habermann, N.: Prevention of System Deadlocks. CACM 12, July 1969.

/44/ Hansen, P.B. (ed).: RC4000 Software Multiprogramming System, A/S Regnecentralen, April 1969, Falkoner Alle 1, Copenhagen F.

45/ Hansen, P.B.: The Nucleus of a Multiprogramming System, CACM 13, April 1970, 238.

/46/ Harrison, M.C.: Implementation of the SHARER2 Time-Sharing System. CACM 11, 12, Dec. 1968, 845.

/47/ Hart, L.E.: The User's Guide to Evaluation Products, Datamation 17, 1, Dec. 1970.

/48/ Hellermann, H.: Digital Computer System Principles. McGraw Hill, 1967. New York.

/49/ Horning, J.J., Randell, B.: Structuring Complex Processes, IBM Research Rpt. RC 2459, May 2, 1969, IBM Research Center, P.O.B. 218 Yorktown Heights, N.Y. 10598.

/50/ IBM. OS/360 Concepts and Facilities. In Programming Languages and Systems. (S.Rosen, Ed.) McGraw-Hill, 1967, 598. New York.

/51/ Iliffe, J.K.: Basic Machine Principles. American Elsevier, 1968. New York.

/52/ Iliffe, J.K., Jodeit, J.G.: A Dynamic Storage Allocation Scheme. The Computer Journal, Oct. 1962, 200-209.

/53/ Jodeit, J.G.: Storage Organization in Programming Systems. CACM 11, 11, Nov. 1968, 741-746.

/54/ Keister, W. et al.: No.1 ESS: System Organization and Objectives. Bell System Technical Journal 43, 5, Sept. 1964, 1831-1844.

/55/ Kilburn, T. et al.: One-Level Storage System. IRE Trans. EC-11, 2, Apr. 1962, 223-235.

/56/ Kleinrock, L.: A Continuum of Time Sharing Algorithms. Proc. AFIPS Conf. 36, 1970 SJCC

/57/ Knuth, D.E.: Additional Comments on a Problem in Concurrent Programming. CACM 9, May 1966. 321-323.

/58/ Lampson, B.W. et al.: A User Machine in a Time-Sharing System. Proc. IEEE 54, 12, Dec. 1966.

/59/ Lampson, B.W.: Dynamic Protection Structures. Proc. AFIPS Conf. 35. 1969 FJCC

/60/ Lampson, B.W.: On Reliable and Extensible Operating Systems. Infotech State of the Art Proceedings, 1970. Maidenhead, England.

/61/ Lanzano, B.C.: Loader Standardization for Overlay Programs. CACM 12, Oct. 1969, 541-550.

/62/ Linde, R.R. et al.: The ADEPT-50 Time Sharing System. Proc. AFIPS Conf. 35, 1969 FJCC

/63/ MacDougall, M.H.: Computer System Simulation: An Introduction. Cs 2,3, Sept. 1970, 191-210.

/64/ Manacher, G.K.: Production and Stabilization of Real-Time Task Schedules. JACM 14, 3, July 1967, 439-465.

/65/ Mattson, R.L. et al.: Evaluation Techniques for Storage Hierarchies. IBM Sys. J. 9, 2, 1970, 78-117.

/66/ McCarthy, J., Corbato, F.J., Daggett, M.M.: The Linking Segment Subprogram Language and Linking Loader. CACM 6, 7, July 1963. 391-395. In Rosen, S. (ed), Programming Languages and Systems, McGraw Hill, 1967. New York.

/67/ McKinney, J.M.: A Survey of Analytical Time-Sharing Models. CS 1, 2, June 1969, 105-116.

/68/ Mealy, G.: The System Design Cycle, Proc. 2SOSP. Oct. 1969.

/69/ Molho, L.: Hardware Aspects of Secure Computing. Proc. AFIPS Conf. 36, 1970, SJCC, 135.

/70/ Morris, R.: Scatter Storage Techniques. CACM 11, 1, Jan. 1968, 38-44.

/71/ Murphy, J.E.: Resource Allocation with Interlock Detection in a Multi-task System. AFIPS Conf. Proc. 33, 1968 FJCC, 1169-1176.

/72/ NATO Report on SOFTWARE ENGINEERING, Garmish, Oct. 1968.

/73/ NATO Report on SOFTWARE ENGINEERING, Rome, Oct. 1969.

/74/ Naur, P. et al.: Revised ALGOL Report CACM 6, 1, Jan 1963, 1-17. In Rosen, S. (ed.) Programming Languages and Systems, McGraw, 1967. New York.

/75/ Pankhurst, R.J.: Program Overlay Techniques. CACM 11, 2, Feb. 1968, 119-125.

/76/ Parkhill, D.: Challenge of the Computer Utility. Addison Wesley, 1966. Reading, Massachusetts.

/77/ Randell, B., Kuehner, C.J.: Dynamic Storage Allocation Systems. CACM 11, 5, May 1968, 297-305.

/78/ Randell, B., Russel, L.J.: ALGOL 60 Implementation. Academic Press, 1964. London.

/79/ Saltzer, J.H.: Traffic Control in a Multiplexed Computer System. MIT Project MAC Rpt MAC-TR-30, 1966. Project MAC, 545, Technology Square, Cambridge, Mass. 02139.

/80/ Saltzer, J., Gintell, J.: The Instrumentation of MULTICS. CACM 13, Aug. 1970.

/81/ Sayre, D.: Is Automatic Folding of Programs Efficient Enough to Displace Manual? CACM 12, 12, Dec. 1969, 656-660.

/82/ Sharpe, W.F.: The Economics of Computers. (Chapt. 10) Columbia University Press, 1969.

/83/ Spier, M.J., Organick, E.I.: The MULTICS Inter-process Communication Facility, Proc. 2SOSP, Oct. 1969.

/84/ Subcommittee of the American Standards Association Sectional Committee X3, Computers and Information Processing. FORTRAN vs. Basic FORTRAN-A Programming Language for Information Processing on Automatic Data Processing Systems. CACM 7, 10 Oct, 1964, 591-624.

/85/ Vanderbilt, D.H.: Controlled Information Sharing in a Computer Utility. MAC-TR-67, MIT, October 1969, Document Room, Project MAC 545 Technology Square, Cambridge, Mass 02139.

/86/ Warner, C.D.: Monitoring: A Key to Cost Efficiency. Datamation 16, 17, Jan. 1971.

/87/ Wegner, P.: Communication Between Independently Translated Blocks. CACM, July 1962, 376-381.

/88/ Wegner, P.: Programming Languages, Information Structures and Machine Organization (Sections 4.1-4.7). McGraw Hill, 1968. New York.

/89/ Weissmann, C.: Security Controls in the ADEPT-50 Time-Sharing System. Proc. AFIPS Conf. 35, 1969 FJCC, 119.

/90/ Wilkes, M.V.: A Model for Core Space Allocation in a Time Sharing System. AFIPS Conf. Proc. 34, 1969 SJCC, 265-271.

/91/ Wilkes, M.V.: Slave Memories and Dynamic Storage Allocation. IEEE Trans. EC-14, Apr. 1965, 270-271.

/92/ Wilkes, M.V.: Time-Sharing Computer Systems. Am. Elsevier, 1968. New York.

/93/ Wirth, N.: On Multiprogramming Machine Coding and Computer Organizations. CACM 12, 9, Sept. 1969, 489-498.

/94/ Wulf, W.A.: Performance Monitors for Multiprogramming Systems, Proc. 2SOSP, Oct. 1969.

/95/ Zurcher, F., Randell, B.: Iterative Multi-Level Modelling. A Methodology for Computer System. IFIP Congress 68, Edinburgh, Scotland, Aug. 1968.

LOHONYAI MIKLÓS

# Távadatfeldolgozás

LEKTORÁLTA:
BOHUS MIKLÓS

# Bevezetés

A távadatfeldolgozás a belátható jövő (és részben már napjaink) távközléstechnikája és számitástechnikája. Mivel két, önmagában is széles spektrumu szakterület ötvözéséről van szó, a távadatfeldolgozás egyes részletkérdéseiről is vaskos köteteket lehetne (és kellene) irni.

E rövid tanulmány "A számitástechnika legujabb eredményei" cimü sorozatba illeszkedik, s igy feladata csupán a távadatfeldolgozó rendszerekre általában jellemző, különösen a távadatfeldolgozásban alkalmazott adatátvitellel kapcsolatos kérdések vizsgálata volt. A rendelkezésre álló terjedelem igy is csak egy általános rendszertechnikai összefoglalót tett lehetővé, amelyben többnyire csupán a felmerülő problémák heurisztikus felvetéséig juthattunk el, és nem foglalkozhattunk számos, a felhasználó szempontjából lényeges és érdekes kérdéssel. Arra sem volt mód, hogy konkrét alkalmazások, módszerek, eljárások kapcsán - mintegy esettanulmányként - részletekbe mélyedjünk.1)

A tanulmány e fogyatékosságai ellenére - mint rendszertechnikai összefoglaló - hiányt pótol, és bizom benne, hogy legalább a távadatfeldolgozással kapcsolatos rendszertechnikai szemlélet és egyértelmü terminológia kialakitásához az olvasónak utmutatást nyujt.

A tanulmány megirásában sokat segitett, hogy az elmult évek során munkahelyemen, a Nemzetközi Számitástechnikai Oktató Központban lehetőségem nyilt -neves hazai és külföldi szakemberek bevonásával - a távadatfeldolgozás témakörében speciális tanfolyamok szervezésére. Ezuton is köszönetet mondok a fenti tanfolyamokra meghivott valamennyi - mindenekelőtt Bohus Miklós, Dr. Gordos Géza, Mazgon Sándor, Németh József és Wollner Róbert - előadónak, valamint Csomor Gyula és Limperger István közvetlen munkatársaimnak, akik előadásaikkal és tanácsaikkal támogatták munkámat. Külön megköszönöm Bohus Miklós lelkiismeretes, minden részletre kiterjedő, alapos szakmai lektori munkáját és hasznos észrevételeit.

A szerző

---

1) Itt utalnék a könyvsorozat többi tanulmányaira. (Pl. Bakos László: "Többszámitógépes rendszerek", Huba Zoltán: "Időosztásos üzemmód" c. müvére.)

# 1.

# A távadatfeldolgozás fejlődési folyamata, általános problémái és alkalmazásai

Az utóbbi évek során a *távközléstechnika* és a *számitástechnika integrálása*, a *távadatfeldolgozás* terén - a számitástechnika egészének gyors fejlődéséhez képest is - *forradalminak tünő változások* tanui vagyunk.

E fejlődés irányvonala ma már egyértelmüen azt mutatja, hogy a sokak által még mindig csak a számitástechnika egy speciális alkalmazásának tekintett távadatfeldolgozás olyan országos, sőt nemzetközi méretü *komplex információs rendszerek* megalkotásában teljesedik ki, amely rendszereknek az egyes számitóközpontok csupán épitőkockái. Az ilyen komplex rendszerekben azonban a számitógép már nemcsak feldolgozó központként müködhet, hanem a kommunikációs hálózat fontos láncszemének szerepét is betöltheti.

Korunkban a müszakilag és gazdaságilag fejlett országokban az ember alkotta *rendszerek* földrajzi kiterjedése, bonyolultsági foka és kapcsolatainak száma igen nagymértékben megnövekedett. A nagy gazdasági - termelési, kereskedelmi és szolgáltató - rendszerek információéhsége és információ feldolgozási igénye csak számitógépekkel elégithető ki. Kezdetben ez a követelmény csak egy-egy vállalat, intézmény vagy szervezet szintjén jelentkezik.

A fejlődés azonban megállithatatlan és a komplex információs rendszerek előbb-utóbb *alapvető társadalmi-gazdasági szükségletté* válnak. Olyan szükségletté, mint amilyen a tömegközlekedés vagy az anyagi javak szállitásának gépesitése volt még nem is olyan régen, és olyanná, amelynek kielégitetlensége a további fejlődés gátja lenne.

A nem tul távoli jövőben a fejlett országokban a komplex információs rendszerek ugyanolyan alapvető nemzetgazdasági értéket és erőforrást képviselnek majd, mint amilyet a legfontosabb iparágak vagy az országos méretü energiaszolgáltatás jelentenek ma.

Elfogadott vélemény, hogy a távközléstechnika és a számitástechnika integrálása olyan *uj információs forradalmat* hiv életre, amelynek jelentősége a könyvnyomtatás vagy a távolbalátás elterjedésének jelentőségéhez hasonlitható.

Ez az uj revolució annyiban jelent minőségi különbséget a korábbiakhoz képest, hogy *az információk keletkezésük helyéről közvetlenül a feldolgozás helyére, a számitógépes feldolgozás eredményei pedig automatikusan rendeltetési helyükre jutnak.* Az információk továbbitása és feldolgozása egymással összefonódott közvetlen kapcsolatban, elektronikus sebességgel, a *tér-* és *időkorlát legyőzésével* történik.

A komplex információs rendszerekben a-*számitástechnika erőforrásai közüzemi szolgáltatássá, közkinccsé válnak.*

A komplex információs rendszerek létrehozása hazánkban ma még a jövő tervei közé tartozik, a felkészülés az ilyen rendszerek kiépitésére, az integrálási folyamat tevékeny végigvitele, a távadatfeldolgozással való foglalkozás azonban már napjaink égetően sürgős feladata. A Számitástechnikai Központi Fejlesztési Program III. szakasza közmüszerü számitástechnikai szolgáltatás megvalósitását irja elő hazánkban.

A távadatfeldolgozás fejlődésének igen gyors ütemét mi sem bizonyitja jobban, mint az, hogy az integrálódás rövid, másfél- két évtizedes folyamata is jól elkülönithető fokozatokra bontható.

(Az alábbi ütemezés elsősorban az Amerikai Egyesült Államokra igaz, a többi országok késéssel követik a fejlődést.)

A hatvanas évek közepéig terjedő időszakot *az integrálódási folyamat elkezdődése,* az évtized második felét *az evolució felgyorsulása,* az évtized legvégét, a hetvenes évek elejét *a revolució kezdete* jellemzi:

1) *Az integrálódási folyamat kezdetének időszakában* a digitális információ elektronikus továbbitásának, az *adatátvitelnek* kidolgozott elmélete (C.E. Shannon, 1948) és gyakorlati módszerei (távíró, telex, modulációs eljárások stb.) voltak.

A számitástechnikát előbb a *soros,* majd a *multiprogramozásos* alapon történő *helyi kötegelt feldolgozás* (batch processing) jellemezte. Nyilvánvalóvá váltak a *magasszintü programozási nyelvek* és az *operációs rendszerek* használatának előnyei. Az adatátvitelt és a számitástechnikát sok rendszerben kezdték közvetett módon, *másodlagos adathordozók* közbeiktatásával összekapcsolni; ezt *off-line távadatfeldolgozásnak* nevezik.

Szinte kizárólag a kutatás szintjén, valamint néhány ipari irányitó-, és ügyviteli információs rendszerben, elsősorban azonban katonai létesitményekben alkalmazták a számitógéphez nagy távolságból *közvetlenül, on-line* kapcsolt berendezéseket. Igy hangolták például össze - *időazonos, real-time módon* - egy földrajzilag nagy területre kiterjedő légvédelmi rendszer müködését (SAGE, BUIC).

A katonai rendszerekhez hasonló elvek szerinti felépült, korai ügyviteli információs rendszer volt az első amerikai repülőgép helyfoglaló rendszer is (SABRE).

2) *Az evolució felgyorsulásának időszakában* számos olyan trend jelent meg a számitástechnikában, amelyek lényegében a mai fejlődés mozgató rugói is. Ezen irányzatok elméleti alapjainak lerakása és a velük kapcsolatos első kisérletek nagyrészt az M.I.T. (Massachusetts Institute of Technology) kutató csoportja által a hatvanas évek első felében elkezdett és ma is folytatott munka eredményei.

Ebben a korszakban általánossá vált a fejlett architekturával és operációs rendszerrel rendelkező, közepes és nagyteljesitményü számitógépek használata. Az *adatátviteli programcsomagok* (pl.

BTAM, QTAM) az operációs rendszerek állandó kisérő rendszerprogramjává váltak.

A hardware-ral megvalósitott *adatátviteli vezérlő egységek* (pl. IBM 2701; 2702; stb.) mind több számitóközpontban bukkantak fel.

Rohamosan növekedett a számitógéptől nagyobb távolságban elhelyezett - adatátviteli vonal használatát igénylő - "perifériális berendezések", a *terminálok* száma és választéka.

A polgári életben is elterjedt a számitógép távolról történő *on-line* felhasználása, először csak mint *távolsági kötegelt adatfeldolgozás* (remote batch processing).

Egy adott számitógép erőforrásait egyre *több felhasználóhoz* (multiuser systems) közvetlenül juttatták el.

Tovább fokozódott az ipari *real-time folyamatirányitó rendszerek* térhóditása.

Ezek mellett széleskörben elterjedtek az olyan ügyviteli feldolgozó rendszerek is, amelyek idősveszteség nélkül, de legalábbis a számitógéppel kiszolgált folyamat időléptékéhez képest gyorsan, igény alapján adják meg a választ az összes, on-line kapcsolt távoli felhasználónak.

Ezeket nevezzük *real-time adatbank orientált rendszereknek.*

Az ilyen rendszerekben már nagy figyelmet forditottak az *ember és a számitógép kapcsolatának* megjavitására.

A termináloknál tevékenykedő felhasználók "párbeszédet" folytathatnak a számitógéppel, az ember és a számitógép ismétlődő információcsere, *interaktiv tevékenységsor* formájában kommunikál egymással.

A felhasználók itt azonban adminisztrátorok, akik vagy szabványos formában informálják a számitógépet, vagy szabványkérdésekkel információkat nyernek a központi adatbankból, tehát egyetlen *központi programrendszeren* dolgoznak. A termináloknál kifejtett funkciók igen korlátozottak, a felhasználók nem generálnak uj programokat és nem oldanak meg jellegében uj problémákat.

A terminálok száma nagy, általában többszáz.

További fejlődést jelentett az evolució felgyorsulásának időszakában az *időosztásos, time-sharing rendszerek* elterjedése.

(Project MAC és MULTICS as M.I.T-nél; QUICKTRAN az IBM-nél stb.)

Az időosztásos rendszerek is rendelkeznek az *időazonosság* képességével és az *interaktivitás* tulajdonságával.

A time-sharing rendszerek azonban többnyire *független felhasználókat* szolgálnak ki, akik egymástól *független problémákon*, különböző programnyelveken dolgoznak és *interaktiv tevékenységsorban* használják a rendszert.

A *felhasználók* nem adminisztrátorok, hanem programozni tudó *probléma-megoldók* - mérnökök, tudományos kutatók, egyetemi hallgatók - vagy programfejlesztők. Számuk legfeljebb néhány tucat. Feladatuk megoldását a számitógéppel távolról folytatott intenziv "párbeszéd" utján, lépésről lépésre keresik meg és a végén közvetlenül megkapják az eredményt.

A programokra nem jellemző az ismételt futtatás.

A problémamegoldó ember és a számitógép kapcsolata nagymértékben megjavul a kötegelt feldolgozáshoz képest. Az ember közvetlenül és azonnal ellenőrizheti ötletei életrevalóságát és a megoldás keresése közben végrehajtott módositások kihatásait. A számitógéppel általában nagyon egyszerü és igy gyorsan elsajátitható nyelven (pl. BASIC) "társalog".

Minden felhasználónak az az illuziója, hogy térben és időben visszakerült a számitógéphez és azt egyedül használja. Az illuzió oka az, hogy minden felhasználóhoz egy látszólagos, *virtuális számitógépet* rendel a rendszer. A virtuális számitógép csak logikailag létezik, fizikailag *a rendszer kijelölt erőforrásait* egy-egy *időszeletre* birtokolja csak minden egyes felhasználó.

Az időszelet lehet fix vagy dinamikusan változó hosszuságu, időtartama azonban elegendő hosszu a számitógép számára egy-egy részprobléma megoldásához és elegendő rövid az ember időléptékéhez képest (pl. 8-10 ms).

A rendszer belső adminisztrációja természetesen viszonylagosan felduzzad, ezért - bár a felhasználó problémáinak megoldása felgyorsul, és az ember kihasználtsági foka megnövekszik - a számitógép kihasználtsága közepes szinten marad. Ezt elkerülendő, a "holtidőben" gyakran a korábban belőtt programnak ismételt, kötegelt feldolgozására használják a rendszert.

A time-sharing rendszer szolgáltatást nyujt a felhasználóknak, eljuttatja azokhoz is a számitástechnika erőforrásait, akik nem engedhetik meg maguknak, hogy önálló számitógépet szerezzenek be.

Az ilyen általános célu time-sharing rendszerek mellett számos speciális célu időosztásos rendszer is megjelent és teret kért magának.

(Tulajdonképpen az időazonos és az időosztásos rendszerek közötti különbség sok esetben elmosódik, igy definiálásukban és elnevezésükben a szakirodalom nem egységes.)

A hatvanas évek utolsó harmadában a második, majd harmadik generációs kisszámitógépek (minicomputerek) száma igen gyors növekedésnek indult, az évtized végére áruk az integrált áramköri technológiák uralkodóvá válása következtében jelentősen lecsökkent. (Ez a folyamat még ma is intenziven tart.) Ennek köszönhető, hogy a központi feldolgozó számitógépet egyre több "intelligens" kisegitő, kisszámitógép vette körül és a távadatfeldolgozó rendszerek felépitése alaposan megváltozott.

A rendszerben flexibilis, programozással módositható tulajdonságokkal rendelkező *adatátviteli számitógépek* jelennek meg, amelyek az adatátvitel vezérlésén tulmenő legkülönbözőbb előfeldolgozási és kisegitő feladatokat is ellátják. Ezek a rendszerben ellátott funkcióik szerint adatátviteli processzorok, front-end processzorok, adatkoncentrátorok, intelligens terminálok lehetnek.

A kisszámitógépek adatátviteli számitógépként való felhasználásának üteme napjainkban is lendületesen tovább fokozódik.

Ezekben a rendszerekben tehát már több számitógép összekapcsolásával jön létre - a komplex rendszerek előfutáraként - *a távadatfeldolgozó hálózat.*

Nyilvánvaló, hogy a fentiekben felsorolt terminál orientált online rendszerek létrehozásához az adatátvitel igen széleskörü alkalmazására volt szükség.

Azonban még a leggazdagabb országok sem engedhetik meg maguknak, hogy a távadatfeldolgozáshoz azonnal uj adatátviteli szolgáltatásokat hozzanak létre. Ezért erre a korszakra az jellemző, hogy a hagyományos távközlési szolgáltatásokat extenziven alkalmazzák az adatátvitel lebonyolitására.

A szabványositott adatátviteli kódok és adatátviteli eljárások széleskörben elterjednek. Elsősorban még alacsony és közepes sebességü adatátvitelt alkalmaznak. A hagyományos postai távközlési rendszereket "feljavitják", megbizhatóságukat, adatátviteli sebességüket fokozzák. Ez számtalan kompromisszumos megoldáshoz vezet.

A nemzetközi távközlési és szabványositási szervezetek (CCITT, I. S. O. stb. [1]) a nagy számitógépgyártó cégekkel karöltve ajánlásokat és szabványokat dolgoznak ki a számitógépek és az adatátviteli hálózat összekapcsolására és együttmüködésére, és ezek használata szinte azonnal átmegy a gyakorlatba.

Az egész időszakra jellemző, hogy a központi számitógépek ára évenként 20-25 %-kal zuhant (azonos teljesitményt feltételezve), de a rendszer összköltsége mégis emelkedett. Az összköltségen belül a software-költség domináló lett.

Uj, jelentős költségtényezők léptek fel: a *terminálrendszer* és *az adatátvitel költségei.*

A terminálok és a hagyományos perifériák ára nem csökkent olyan mértékben, mint a központi számitógépé. A terminálok többsége olcsó a perifériákhoz képest, számuk azonban nagy volt.

Az adatátviteli költségek stagnálnak, a költségcsökkentés utjai rögösek. A legjárhatóbbnak tünő ut az *adatátviteli hálózatok optimalizálásának és az adatvonalak többszörös kihasználásának módszere,* amelyre a hardware berendezések mellett bevetették a kisszámitógépeket. Ehhez társult még a meglévő adatvonalak átviteli sebességének növelése, amely csak a közepes sebességtartomány felső határáig eredményez - messze a sebességnöveléssel arányosnál alacsonyabb mértékü - költségcsökkenést.

Már ebben a korszakban is müködött több olyan távadatfeldolgozó rendszer, amelyben *több* - rendszerint egy gépcsaládba tartozó -

---

[1] CCITT: Comité Consultatif International Télégraphique et Téléphonique, Nemzetközi Távíró és Távbeszélő Tanácsadó Testület; I. S. O.: International Organization for Standardization, Nemzetközi Szabványügyi Szervezet.

*központi számitógépet* egyetlen rendszerben *kapcsoltak össze* (pl. PARS, GE Information Services Network stb.)

3) A *revolució elkezdődése* egyáltalán nem volt véletlen és váratlan. A hatvanas évek végére teljességgel igazolódott, hogy a távközléstechnika és a számitástechnika "házassága" sikeres, életképes és termékeny.

Ugyanakkor az alapvetően egyetlen számitóközpontra épitett terminál-orientált real-time és time-sharing rendszerek átestek a gyakorlat próbáján és felfedték korlátaikat.

Szükségszerüen bekövetkezett tehát az e rendszerekben alkalmazott *elvek* olyan rugalmas *általánositása*, amelynek realizálása már nagyrészt mentes az elődök hibáitól.

Az elődök hibáinak forrása *autarchiájuk*, az a tény, hogy csak viszonylag szük felhasználói közösség rendelkezik az általuk nyujtott szolgáltatásokkal.

Ennek következtében *az egyes rendszerek legjelentősebb beruházási tételei*, a rendszer és felhasználói software, valamint az applikáció költségei minden egyes rendszerben ismételten, tehát társadalmi szinten *multiplikáltan jelennek meg*. Egy-egy kidolgozott probléma-megoldó képességnek egy másik autarchikus rendszerben történő reprodukálása az egyes rendszerek különbözősége miatt komoly nehézségeket, hosszu időt és nagy kiadásokat jelent.

A számitógépes szakemberek tömegét kell átprogramozással, az adat-file-ok átszervezésével foglalkoztatni.

E tények felismerése alapján szinte nyilvánvaló az az igény, hogy az egyes önellátó számitástechnikai előfizetői csoportosulások szolgáltatási és területi széttagoltságát megszüntető, *komplex számitástechnikai hálózatokat* hozzunk létre.

Az ilyen komplex többszámitógépes hálózat olyan autonóm távadatfeldolgozó rendszerek összessége, amelyben az egyes rendszeret tetszőleges párositásban, interaktivan részesedhetnek egymás erőforrásaiból. Bármely előfizetőnek a "saját" *központi számitógépe* szolgáltatásán tulmenően, *magasabb szintü rendszer-szolgáltatás* is nyujtható.

A számitástechnika erőforrásai *közüzemi szolgáltatásként* jelennek meg, ezt nevezik *resource sharingnek*, *erőforrás megosztásnak*.

A resource-sharing hálózatokban általános, hogy az egyes központi számitógépek nem közvetlenül, hanem egy kisebb számitógépek bevonásával realizált nagysebességü adatátviteli hálózat, - ugynevezett *magasszintü hálózat* - utján kommunikálnak egymással.

A hetvenes évek egyik jellemzője éppen az ilyen uj, kimondottan számitógépes adatátvitel céljára létrehozott nagysebességü digitális adatátviteli hálózatok megjelenése, és általuk a számitógépek országos (sőt nemzetközi) méretekben történő összekapcsolása (CYBERNET, ARPANET, NPL, CYCLADES stb.).

Ugyancsak jellemző a számitógépek közötti kommunikációban a távközlési müholdak részvétele.

Természetesen a hagyományos távközlési hálózatok alkalmazása ma még dominál és a komplex rendszerek termináljai is többnyire hagyományos módon kapcsolódnak központi számitógépükhöz.

A ma müködő komplex rendszerek többségének hibája a nagymértékü heterogenitás. Az utóbbi időben azonban kezdenek előtérbe kerülni a nagyfoku szabványositásra, a hardware-software kompatibilitásra és uniformitásra irányuló törekvések.

Az Amerikai Egyesült Államokban létrehozott komplex számitástechnikai hálózatok is többfélék - a módszerek még nem egyértelmüen kidolgozottak -, és nem adaptálhatók kritika és a körülmények, igények, a nemzeti-gazdasági sajátosságok figyelembevétele nélkül.

Sok olyan vélemény is létezik, amelyek szerint elsősorban az olyan, kisszámitógépekre épitett távadatfeldolgozó hálózatok létrehozása kivánatos, amelyekben legfeljebb néhány közepes vagy nagy központi feldolgozó számitógép müködik.

Abban azonban csaknem minden szakember egyetért, hogy előbb-utóbb olyan komplex, távadatfeldolgozáson alapuló, információs rendszereket kell létrehozni, amelyek könnyen hozzáférhető szolgáltatást nyujtanak előfizetőiknek.

A távadatfeldolgozás fejlődési folyamata rövid összefoglalásának lezárásaként a multat, a jelent és a közeli jövőt nézzük meg a statisztikák és előrejelzések tükrében is.

1. táblázat

A terminálok számának alakulása néhány európai országban

| Ország | 1972 | 1976 | 1980 | 1985 |
|---|---|---|---|---|
| | Ezer | | | |
| Német Szövetségi Köztársaság | 14,6 (18,0) | 51,5 (81,1) | 105,0 | 220,5 |
| Franciaország | 11,0 (14,0) | 33,9 (61,0) | 66,2 | 135,3 |
| Anglia (Egyesült Királyság) | 36,3 (31,0) | 57,5 (86,1) | 107,8 | 193,1 |
| Svájc | 2,2 (2,3) | 8,9 (9,0) | 16,8 | 26,2 |
| Összesen[1] | 79,6 (86,6) | 235,6 (325,5) | 437,4 | 815,0 |

---

[1] A táblázat kivonat az "Eurodata" tagországaira kiterjedő felmérésből /Irod. 1. /, az összesités 17 európai országra vonatkozik. A számadatok nem tartalmazzák a "helyi" terminálok számát. Az 1972-es és 1976-os oszlopokban zárójelben feltüntetett adatok a Logica Ltd. előrejelzései, amelyek a "helyi" terminálokat is magukba foglalják. (Logica Ltd. előadás. SZÁMOK, Távadatfeldolgozás c. speciális tanfolyama, 1973. szeptember, Balatonkenese.)

2. táblázat 1)
A terminálok számának alakulása
az Egyesült Államokban

| Ország | 1970 | 1974 | 1980 |
|---|---|---|---|
| Egyesült Államok | 185 ezer<br>(300 ezer) | 820 ezer | 2 425 ezer<br>(4 000 ezer) |

3. táblázat
A számitástechnika-távadatfeldolgozás beruházási
költségaránya

| | 1955 | 1965 | 1975 | 1985 |
|---|---|---|---|---|
| Központi hardware | 50% | 35% | 10% | 5% |
| Üzemeltetés | 25% | 30% | 30% | 20% |
| Software | 25% | 35% | 30% | 25% |
| Adathálózat, adatállomás | - | - | 30% | 50% |

***A távközléstechnika és a számitástechnika integrálásának alapvető problémája***, hogy ***két különböző müszaki és gazdasági*** paraméterekkel rendelkező ***rendszert*** kell egyesiteni.

A távközléstechnikát nagy tehetetlenségü, lassan megváltoztatható és lassan megtérülő, nagy beruházások jellemzik. A távközlésben domináló tényezők: az alacsony sebesség, a perc nagyságrendü összeköttetés és a nagy adatátviteli hibaarány. A távközlés általában hagyományos, jól szabványositott eljárásokat és berendezéseket alkalmazó és bárki által hozzáférhető "nyilt üzlet".

A számitástechnika ismérvei ezzel szemben: az ugrásszerü fejlődés, a gyors elavulás, a nagy sebesség, az összeköttetési idők igen széles spektruma és a nagy megbizhatóság. A számitástechnika eljárásai és berendezései rendkivül heterogén halmazt képeznek és általánosnak tekinthető a számitástechnika "zárt üzlet" jellege is.

A két rendszer egyesitése mindkettő fejlődésének igen nagy lendületet adott és meggyorsitotta mindkét rendszer hibáinak kiküszöbölését.

Az eredőként létrejött távadatfeldolgozó rendszerek általános tulajdonságai a következők:

- a rendszer heterogén, ez egyaránt igaz az alkalmazott számitástechnikai berendezések hardware és software tulajdonsá-

---

1) Az Egyesült Államokra vonatkozó adatok a Logica Ltd.-től származnak /Irod. 2. /, a zárójelben feltüntetett értékek forrása a Datamation /Irod. 3. /. Ez utóbbi szerint a modemek száma az Egyesült Államokban 1970-ben 480 ezer, és ez 1980-ra megtizszereződik.

gaira, az adatátviteli eljárásokra, az adatátviteli hálózatok realizálására - hogy csak a legfontosabb tényezőket emlitsük;

- a rendszer erőforrásain sok felhasználó osztozkodik, a felhasználók versengenek az erőforrások használatáért;
- a felhasználók többsége távolról veszi igénybe a rendszert, "véletlenszerüen" jelentkezik be, gyakran igényel választ és gyors kiszolgálást vár a rendszertől;
- az adatátvitel időigényes, az adatforgalom megjósolhatatlan, csucsterhelések léphetnek fel.

Ezek a többszörösen ellentmondó körülmények és követelmények igy csak kompromisszumok árán hozhatók egyensulyba.

A felsorolt általános jellegü nehézségek mellett az egyes konkrét rendszerekben számos további probléma vár megoldásra.

A távadatfeldolgozó rendszereket különböző szempontok szerint osztályozhatjuk, itt most megkiséreljük, hogy a felhasználónak nyujtott szolgáltatás szemszögéből képezzünk csoportokat. Eszerint beszélhetünk:

a) speciális célu rendszerektől és
b) általános célu munkabeviteli rendszerekről.

a) *Speciális célu rendszerek*
1) adatgyüjtő - adatszétosztó rendszerek,
2) lekérdező rendszerek,
3) dedikált time-sharing rendszerek,
4) folyamatirányitó rendszerek,
5) üzenetkapcsoló rendszerek,
6) egyéb különleges rendszerek.

- a. 1) *Az adatgyüjtő (acquisition) és az adatszétosztó (retailing) rendszerek* szétválasztva is léteznek, bár igen gyakori együttes alkalmazásuk. Ezekben a rendszerekben még általános az off-line távadatfeldolgozás, a gyorsaság nem elsődleges követelmény. A rendszerben a visszacsatolás az emberen keresztül történik. A terminálokon kifejthető funkciók korlátozottak. Az on-line kapcsolt terminálokon is alacsony az interaktivitás foka. Tipikus példa: nagy intézmények, szervezetek távkoordinálása, a termelésirányitási, készletgazdálkodási rendszerek automatizálásának első fázisa.
- a. 2) *A lekérdező (inquiry) rendszerek* arra szolgálnak, hogy a felhasználók egy központi adatbázisból szelektiven, gyorsan és interaktiv módon releváns információkat nyerjenek. A felhasználók uj programokat nem generálnak, a rendszer általában időazonos müködésü. Tipikus példák: a helyfoglaló, a bank és biztositási ügyleteket lebonyolitó rendszerek.

  Adatgyüjtéssel és adatterjesztéssel kombinálva előnyösek termelésirányitási, készletgazdálkodási stb. feladatok megoldására is (mint az ilyenek automatizálásának második fázisa).

- a. 3) A *dedikált time-sharing* (dedicated time-sharing) *rendszerek* speciális problémák egyetlen (vagy kevés) programnyelven - pl. BASIC, APL, QUICKTRAN - történő megoldását szolgálják. Realizálásukhoz kis-, vagy közepes teljesitményü számitógépet használnak, a rendszereket magasfoku interaktivitás jellemzi.
- a. 4) A *folyamatirányitó* (process control) *rendszerek* ipari, laboratóriumi folyamatok mérésadat-gyüjtését, méréspontellenőrzését és zárthurku szabályozását végzik real-time módon.
- a. 5) Az *üzenetkapcsoló* (message switching) *rendszerek* önmagukban is léteznek, mint "kapcsolt" távközlési hálózatok (például távirattovábbitó rendszerek), de alkalmazhatók ugy is, mint egy felettük álló számitógépes rendszer "intelligens" adatátviteli hálózata. Az üzenetkapcsoló rendszerekben alapvetően nem feldolgozás, hanem csupán az adatállomások által szigoru protokoll szerint összeállitott üzenetek továbbitása történik. Az adatátviteli rendszer csak az üzenetek ellenőrzését, ismétlését, felujitását, naplózását, statisztikai feldolgozását, terjesztését és kisegitő feldolgozását (például kódkonverzió) végzi.
- a. 6) Az *egyéb különleges rendszerek* csoportjába tartoznak a ritkábban előforduló és nehezen besorolható esetek. Ilyenek lehetnek például a szimulációs, a szövegfeldolgozó, az oktató, a rakétairányitó stb. rendszerek.

b) *Általános célu munkabeviteli rendszerek*

1) Távolsági kötegelt feldolgozó (remote batch processing) rendszerek.

2) Általános célu időosztásos munkabeviteli (general purpose conversational time-sharing) rendszerek.

3) Komplex számitástechnikai szolgáltató hálózatok.

Mivel ezek a rendszerek általános problémamegoldásra hivatottak, nem osztályozhatók a problémák jellege, hanem csupán a felhasználó és a rendszer kapcsolata, a problémamegoldás jellege szerint.

- b. 1) Az adagolt feldolgozás tulajdonságai jól ismertek. Az ilyen rendszerek leglényegesebb tulajdonsága és egyben hátránya, hogy a programozó "elszakad" a programjától. Távolsági adagolt feldolgozás esetén az "időbeli elszakadás" még jobban megnövekedhet az időigényes adatátvitel és a csucsterhelések miatt.
- b. 2)-3) Az általános célu time-sharing rendszerek és a komplex számitástechnikai hálózatok legfontosabb ismérveit a

fejezet elején, az integrálási folyamat vizsgálatakor már megbeszéltük. Ezek a rendszerek nagymértékben megjavitják a problémamegoldó ember és a számitógép kapcsolatát.

A továbbiakban elsősorban az evolució felgyorsulásának időszakára jellemző, on-line terminál-orientált távadatfeldolgozó rendszerek tulajdonságaival foglalkozunk, és csak röviden érintjük a revolució korszakának kérdéseit.

# 2.

# A távadatfeldolgozó rendszer elemei

Nem kivánok ismert alapfogalmakat felsorolni - mégis ugy gondolom, hogy az egyértelmü elnevezések hiánya miatt célszerü röviden összefoglalni e dolgozat egységre törekvő terminológiáját. Természetesen itt csak a távadatfeldolgozással kapcsolatos *alapvető rendszerelemekre* térünk ki, de további fogalmakat is értelmezünk majd a többi fejezetben, a részletesebb vizsgálat során.

Egy távadatfeldolgozó rendszer *fizikailag adatátviteli hálózatból* és *adatállomásokból* áll; ez első közelitésben megfelel a két integrált szakterületnek.

Az - *adatátviteli hálózat* (vagy egyszerüen adathálózat) digitális információ továbbitására alkalmas távközlési hálózat, távolsági adatátvitelt biztositó közeg: légvezetékek, kábelek, rádió összeköttetések, távközlési müholdak, valamint az ezekhez tartozó kapcsoló központok, átalakitók és egyéb adatátviteli berendezések összessége.

Az információ továbbitását elemi, bitsoros müködésü *adatcsatornák* végzik. Ezeket az adatcsatornákat néha közvetlenül, általában azonban tér-, idő-, vagy frekvenciaosztás utján *adatvonalak* realizálják.

Az adatállomások közötti *összeköttetések* lehetnek *közvetlenek*, *felfüzésesek*, az összeköttetések egy részét azonban *kapcsoló központok* (KK) hozzák létre több független adatvonal szakasz átmeneti egymáshoz rendelésével (1. ábra).

A kapcsoló központok "kapcsolásmódja" tekintetében megkülönböztetünk *vonalkapcsolást* (line switching) és *üzenetkapcsolást* (message switching). Vonalkapcsolás esetén, a telefonközpontok megszokott módján, közvetlen fizikai kapcsolat jön létre az adatcserében résztvevő adatállomások között.

Üzenetkapcsolásnál ezzel szemben az átvinni kivánt információt vagy ennek egymás után következő részhalmazait átmenetileg tároljuk és csupán később továbbitjuk.

Mindkét fajta kapcsolóközpont vezérléséhez alkalmazhatunk számitógépeket és általában mindkét esetben mód van több lehetséges utvonal közül történő választásra.

Az *adatállomások* lehetnek *számitógépes adatállomások* és *adatvégállomások*.

Azokat az adatállomásokat, amelyeken az adatátviteli hálózathoz on-line kapcsolt, felhasználói problémákat megoldó, *feldolgozó számitógép* müködik, *számitógépes adatállomásnak* (SzGA) nevezzük.

A távadatfeldolgozó rendszerben több számitógépes adatállomás is lehet, valamennyin együttmüködhet egynél több számitógép is. Egy kiválasztott számitógép a saját közvetlen felhasználói köre számára mint *központi számitógép* jelenik meg. Egynél több feldolgozó központ esetében az egyes "központi számitógépek" csak saját felhasználói körük számára "központiak", ilyenkor ezeket *honos számitógépnek* (host computer) nevezzük. A továbbiakban - ha külön nem állitjuk az ellenkezőjét -, az egyszerüség kedvéért ugy tekintjük, hogy a rendszer egyetlen központi számitógépet foglal magába.

A távadatfeldolgozás hatékonyságát igen nagy mértékben befolyásolják a központi számitógép hardware és software tulajdonságai.

A központi számitógéphez lokális háttértárak és általában helyi input-output perifériák is tartoznak.

Az adatátviteli hálózathoz valamelyik input-output csatorna (általában multiplexer, byte-multiplex csatorna) közvetitésével kapcsolódik a központi számitógép.

Az input-output csatorna *adatátviteli vezérlő egységet* (AVE) (Communications control unit) müködtet. Ennek feladatait egy kisszámitógép is elláthatja. Az adatátviteli vezérlő egység általában időmultiplex müködésü berendezés, amely szimultán több adatvonalon is vezérelheti az adatátvitelt.

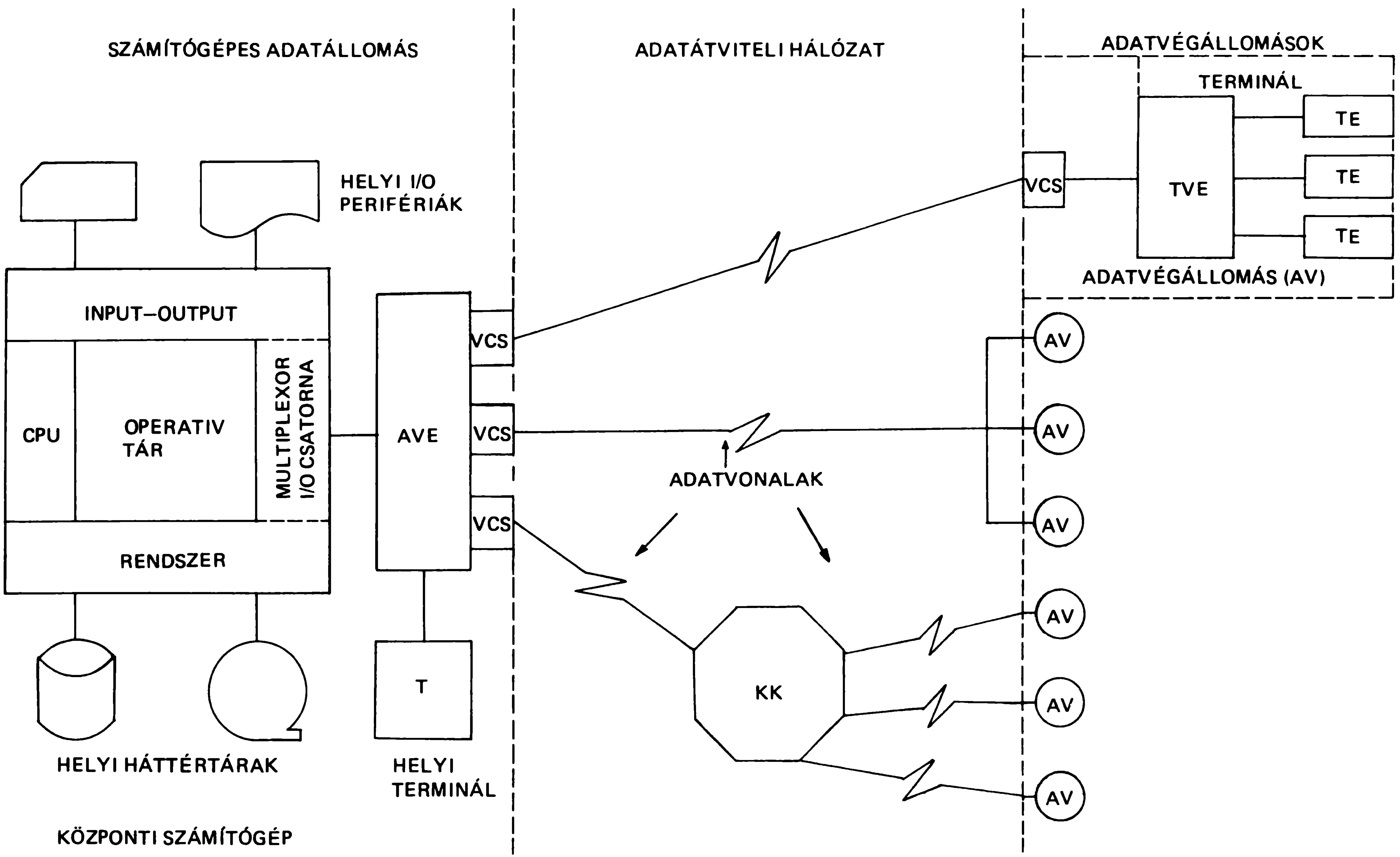
Az adatátviteli hálózathoz fizikailag vonalcsatlakozók (VCS) illesztik a számitógépes adatállomást. Az analóg átvitelt illesztő vonalcsatlakozó neve *modem (mo*dulátor-*dem*odulátor).

A távadatfeldolgozó rendszer és a "távoli környezet" kapcsolata az *adatvégállomásokon* (AV) jön létre. Az adatvégállomás ugyancsak *vonalcsatlakozóval* illeszkedik az adatátviteli hálózathoz.

Az adatvégállomás perifériális berendezése az "adatvégberendezés", a *terminál* (T), amely *terminál vezérlő egységből* (TVE) és egy vagy több terminál input-output egységből vagy egyszerüen *terminál egységből* (TE) áll. A terminál egységek vezérlésének - elsősorban adatátviteli vezérlésének - feladatát szintén bizhatjuk kisszámitógépre.

A terminálok egy része olyan közel is elhelyezkedhet a számitógépes adatállomáshoz, hogy célszerü közvetlen, vonalcsatlakozók nélküli, esetleg párhuzamos átvitellel történő müködtetésük (1. ábra). Ezekben a helyi vagy körzeti, kis kiterjedésü un. telephelyi (inplant) rendszerekben speciális technikákat alkalmaznak, amelyekre csak röviden utalunk. A felsorolt fizikai elemek 1. ábra szerinti összekapcsolásának célja az, hogy az információ feldolgozásának erőforrásai a távoli felhasználók számára közvetlenül hozzáférhetők legyenek. Kommunikáció létrejöhet mindenekelőtt a felhasználók és a feldolgozó központ(ok) között, de az egyes feldolgozó központok között és az egyes felhasználók között is.

Az adatátvitel során továbbitott digitális információ egyaránt lehet program, feldolgozásra váró adat, eredmény és vezérlő jel. (Ezért, ha az adatátvitellel kapcsolatban használjuk az "adat"


SZÁMÍTÓGÉPES ADATÁLLOMÁS
ADATÁTVITELI HÁLÓZAT
ADATVÉGÁLLOMÁSOK
TERMINÁL
TE
TE
TE
TVE
VCS
ADATVÉGÁLLOMÁS (AV)
HELYI I/O PERIFÉRIÁK
INPUT–OUTPUT
CPU
OPERATIV TÁR
MULTIPLEXOR I/O CSATORNA
RENDSZER
AVE
VCS
VCS
VCS
ADATVONALAK
AV
AV
AV
AV
AV
AV
KK
T
HELYI HÁTTÉRTÁRAK
HELYI TERMINÁL
KÖZPONTI SZÁMÍTÓGÉP

*1. ábra*
*A távadatfeldolgozó rendszer fizikai elemei*

szót, akkor általános értelemben vett digitális információt értünk rajta.)

A kommunikáció lebonyolitásához a rendszerben a fizikai elemek mellett *"logikai elemekre"* is szükség van.

A legfontosabb logikai elemek az *üzenet* (message) és az *adatátviteli software.*

Az információt az adatátvitelhez fel kell bontani, a felbontás és átvitel legnagyobb egysége az *üzenet.* Minden üzenetben járulékos információkkal egészitjük ki a ténylegesen átvinni kivánt hasznos adathalmazt. A járulékos információ vezérlési feladatokat lát el. Általában jól elkülönülő *fejléc* (header), de legalábbis vezérlő jelek formájában adjuk meg. Az üzenet hasznos része a *szöveg* (text).

Az üzenetek továbbitása valamilyen *adatátviteli vezérlési eljárás* (line control procedure) szerint történik. Ezt a folyamatot az információcserét lebonyolitó adatállomások között *adatátviteli vezérlő és azonositó karakterekkel folytatott "párbeszéd",* kölcsönös "kézrázás" (handshaking) jellemzi (ld. 6. fejezet).

Az üzenetek összerakását, szétválasztását, átalakitását, sorbaállitását, valamint több adatvonalon történő szimultán átvitelének logikai vezérlését, és ezeknek a folyamatokban az operációs rendszer egészéhez való illesztését az *adatátviteli software* vezérli, illetve végzi. Ez a programcsomag a központi számitógép operációs rendszerének része. (Communication subsystem.)

Természetesen emellett minden felhasználónak - vagy a felhasználók egy-egy csoportjának közösen - a saját alkalmazásához illeszkedő *felhasználói programja(i)* és *adatfile-ja(i)* van(nak).

Ezek felépitése és használata a sok felhasználó jelenléte, valamint az adatátviteli hálózat igénybevétele miatt lényegesen eltérhet a helyi feldolgozásban megszokottól.

Azokban a távadatfeldolgozó rendszerekben, amelyekben a "gépi intelligenciát" *adatátviteli számitógépek* alkalmazásával *elosztjuk a rendszer mentén,* elsősorban az adatátviteli softwaret "osztjuk meg", bár a felhasználónak nyujtott többi szolgáltatások egy részét is a feldolgozó központon kivülre helyezhetjük át.

# 3.

# Az adatvonalak tulajdonságai

Az adatátviteli hálózat "ágait" adatvonalakkal realizáljuk. A ma müködő távadatfeldolgozó rendszerek többségében a hagyományos adatvonalak (telexvonalak, távbeszélő vonalak) alkalmazása domináló, bár egyre több uj adatátviteli szolgáltatás jelenik meg világszerte. Az uj adatvonalak megbizhatósága nagyobb és nagyobb adatátviteli sebességre képesek, mint elődeik. Egyetlen adatvonal egy, vagy több elemi adatcsatornát valósithat meg. Az egyes adatvonalak tulajdonságai lényegesen befolyásolják a teljes adatátviteli hálózat felépitését, szolgáltatásainak minőségét és a csatlakoztatható adatállomások tulajdonságait és illesztését is.

Az adatvonalak "egyesitésével" nyert adatátviteli hálózatok jellemzőit a következő fejezetben vizsgáljuk meg.

## 3.1. *Az adatvonalak átviteli sebessége*

Az adatvonalak *transzfer karakterisztikája* véges, a "vonalállapotok" (bitek, dibitek) közötti interferencia elkerülésének alapvető előfeltétele, hogy a másodpercenként 2 $f_s$ számu szimbólum átviteléhez legalább $f_s$ Hertz sávszélességü adatvonalunk legyen (Nyquist-határ).

Ez tulajdonképpen az átviteli sebesség korlátozását jelenti.

*Adatátviteli sebességük* szerint az adatvonalakat a következő csoportokba sorolhatjuk:

a) *Kissebességü* (keskenysávu) vonalak, 300 bit/s átviteli sebesség alatt. Gyakorlatilag a legkisebb használt "gépi" sebesség 45 bit/s bár adatátvitel természetesen "akármilyen lassan" lehetséges. Nagyon sokszor a középsebességü *távbeszélő csatornát*, a *beszédcsatornát* "osztják fel" több, kissebességü adatcsatornára.

b) *Középsebességü* (hangfrekvenciás) vonalak, 300-9600 bit/s adatátviteli sebességgel. Körülbelül 9600 bit/s az a sebességérték, amely egy nem kapcsolt távbeszélő csatornán különleges technikai megoldásokkal még elérhető.

c) *Nagysebességü* (szélessávu) vonalak, 9600 bit/s feletti adatátviteli sebességgel. Igen gyakori a 48 Kbit/s sebesség. A felső határ egyre jobban kitolódik, az amerikai TELPAK-szolgálat például 240 Kbit/s, sőt 0,5 Mbit/s és 1Mbit/s sebességü vonalakat is üzemeltet. A mikrohullámu, csőtápvonalas és lézersugaras összeköttetésekkel ezeknél több nagyságrenddel nagyobb átviteli sebesség is elérhető.

Egy beszédcsatorna sávszélesség-igényén - bár a telefonvonalak átviteli tartománya 300 Hertz és 3400 Hertz közé esik - definiciószerüen 4 kHz-et értünk. Sokszor a nagysebességü vonalak sebességét az ekvivalens beszédcsatornák számával adják meg. (Például 48 kHz = 12 beszédcsatorna.)

A gyakorlatban az adatvonalakra (és a vonalcsatlakozókra is) - a távirótechnikából eredően - igen gyakran az egy másodperc alatt átvihető *elemi vonalállapotok számával* adják meg az elérhető maximális adatátviteli sebességet. Ha az átvihető legrövidebb vonalállapot "T" szekundum időtartamu, akkor az adatátviteli sebesség:

$$S = 1/T \ /\text{baud}/$$

Egyetlen vonalállapot egy bitnél több információt is hordozhat, ezért a bit/s-ban mért adatátviteli sebesség csak bináris átvitelnél egyezik meg a "baud"-ban mért sebességgel.

$$S \ /\text{bit/s}/ = \log_2 n \cdot S \ /\text{baud}/$$ [1]

Az átvitelhez szükséges frekvencia sávszélesség meghatározásához a "baud"-ban megadott érték irányadó, ezért ezt modulációs sebességnek is nevezik.

## *3.2. Vonali üzemmódok*[2]

A *vonalak üzemmódja* szempontjából nem csupán a hasznos adatok, hanem az adatáramlást kisérő és vezérlő összes jelek mozgásirányát is figyelembe vesszük a vonalak minősitésénél. A vonali üzemmódok nem annyira a vonaltól magától, mint inkább a hozzácsatlakozó adatállomástól függnek. Tulajdonképpen a vonal és az általa összekötött adatállomások együttesen szimplex, félduplex, illetve duplex *áramkört* hoznak létre, ennek ellenére szimplex, félduplex és duplex vonalakról szoktunk beszélni.

a) A *szimplex vonalon* egyirányu információáramlás történik. A szimplex vonalak viszonylag kevéssé terjedtek el adatátviteli célokra, mivel - bár az gyakran előfordul, hogy a hasznos adatok áramlása mindig azonos irányu - általában szükség van a vezérlő jelek ellenkező irányu továbbitására is, hogy az adóállomást informálhassuk a vevőállomás állapotáról, a vétel hibátlanságáról stb. A szimplex vonal egyszerü, *kéthuzalos összeköttetéssel* megvalósitható.

Néha a szimplex vonalat egy jóval lassubb, ellenkező irányu "felügyeleti csatornával" egészitik ki.

---

[1] "n" a vonalállapotok lehetséges értékeinek száma, ennek kettes alapu logaritmusa megadja, hogy egy vonalállapot hány bit információt hordoz.

[2] A szimplex, félduplex és duplex (áramkör, vonal, üzemmód stb.) fogalmának nemzetközi távközlési szervektől származó definiciói és a szakirodalom értelmezései több pontban eltérőek. (A szerző.)

Ilyenkor azonban szigoruan értelmezve már félduplex vagy duplex vonali üzemmódot kapunk, habár a hasznos információ csak egyirányban halad. (Az ilyen vonalhoz csatlakozó modemet azonban szimplexnek nevezik.)

b) A *félduplex vonalak* felváltve küldnek mindkét irányban üzeneteket és vezérlő jeleket is, leggyakrabban - de nem feltétlenül mindkét irányban azonos sebességgel. Különösen középsebességü átviteli rendszerekben terjedtek el nagymértékben. Kiválóan alkalmazhatók interaktiv ember-gép kapcsolat esetén. Előnyük az egyszerü, *kéthuzalos összeköttetés*.

c) A *duplex vonalak* mindkét irányban egyidejüleg vihetik át a hasznos adatokat és a vezérlő jeleket. Kis sebességeknél megoldható az átvitel egyetlen vezetékpárral is, frekvenciaosztás segitségével. Közép- és nagy sebességek esetén általában *négyhuzalos összeköttetést* alkalmaznak.

A gyakorlatban a *kétirányu, egyidejü üzenetáramlás* lehetőségét sokszor nehéz kihasználni, mivel nem áll a vonal mindkét végén egyszerre rendelkezésre szöveg, és mivel általában mindkét irányu adatátvitel egymástól függetlenül válaszjeleket is igényel.

## *3.3. A továbbitott információ felbontása*

Az adatátvitel során az információ mindig valamilyen formában felbontásra kerül, és ezeket a felbontási egységeket - biteket, karaktereket, szavakat, blokkokat stb. - időben egymás után adjuk a vonalra.

Ha az egyes karakterek bitjeit egyszerre, de magukat a karaktereket egymás után visszük át, akkor *bit-parallel* vagy *párhuzamos átvitelről* beszélünk.

A párhuzamos átvitelhez nyilvánvalóan az egyszerre átvinni kivánt bitek számával megegyező számu elemi csatornára van szükség. Ha minden elemi csatornát külön vonallal realizálunk, ez hosszu, lassu vonalak esetében tulságosan megemeli az adathálózat fajlagos költségeit, ezért nagyobb távolságu, kissebességü átvitelre csak ritkán használják a parallel átvitelt.

Néhány száz méteres, esetleg néhány kilométeres körzetben az egyszerü egyenáramu párhuzamos adatátvitel gazdaságos lehet, mivel a terminálok felépitése egyszerübbé válik, hiszen nincs szükség a párhuzamos-soros és soros-párhuzamos átalakitásra.

Egy másik, valamivel költségesebb adatállomási berendezéseket igénylő megoldás az egyszerre több vivőfrekvencián történő bit-parallel átvitel. Itt a leggyakrabban használt eljárás a következő: három frekvenciacsoport (A; B; C) mindegyike négy-négy diszkrét frekvenciát (1; 2; 3; 4) tartalmaz; mindhárom csoportból egyszerre egy frekvenciát $4^3 = 64$ féleképpen választhatunk ki, tehát 64 féle karakter közül bármelyiket a vonalra adhatjuk a kódjának megfelelő három frekvencia (pl. A2; B3; C1) kiválasztásával.

Tágabb értelemben véve a többszintü modulációs eljárásoknál (ld. később) is párhuzamos adatátvitel történik, itt azonban a vonalcsatlakozó berendezés gondoskodik a soros-párhuzamos és párhuzamos-soros átalaiitásról, ezért az átvitel bit-sorosnak tünik.

A gyakorlati rendszerekben ma sokkal nagyobb jelentősége van a *bit-soros* vagy egyszerüen *soros adatátvitelnek*, mint a párhuzamosnak. A soros átvitel háromféle módon valósitható meg: ezek az *aszinkron átvitel*, a *szinkron átvitel* és a kettő keverésének tekinthető *impulzus-folyam átvitel*.

Bár elvi akadálya nincs, hogy a fenti módozatokat parallel átvitelnél is használjuk, gyakorlatilag csak a *párhuzamos szinkron* átvitellel találkozhatunk.

### a) *Aszinkron átvitel*

Az aszinkron átvitel során az információt 1-1 karaktert hordozó "rövid blokkokban" visszük át. Mindenegyes karakterhez tartozó bitsorozatot egy START-jel előz meg és egy (vagy több) STOP-jel zár le, ezért az aszinkron átvitelt START-STOP átvitelnek is nevezik.

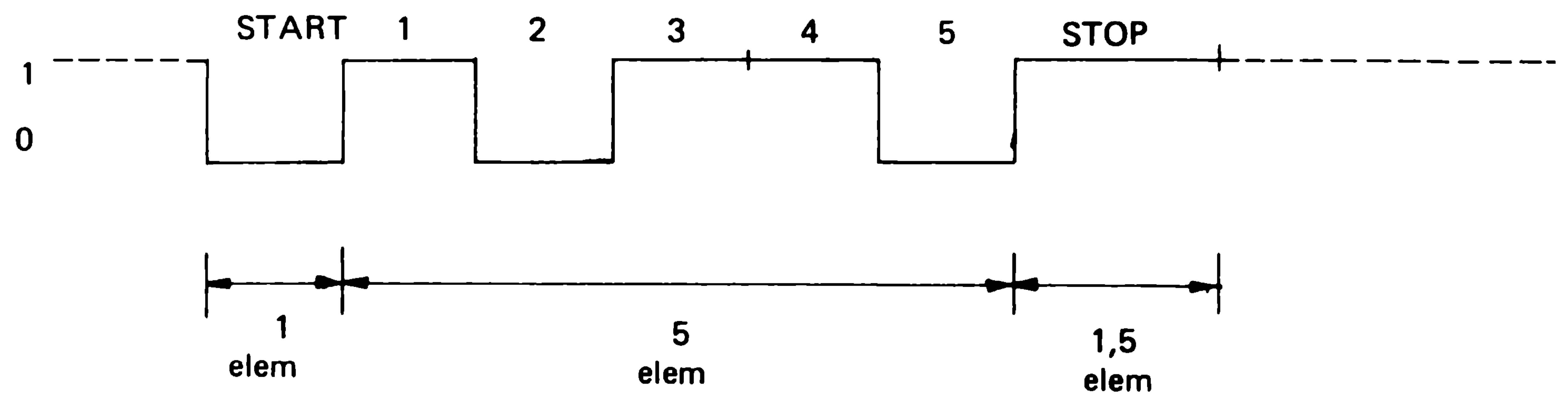
Az egyes karakterek kiadása között - egy türelmi időn belül - tetszőleges idő telhet el. Az adó és vevő adatállomások csupán a START és STOP jelek közötti időintervallumban vannak egymáshoz szinkronizálva.

A vevőállomás "ismeri" az adási frekvenciát, és igy a START-bit hatására elinduló vevőoldal "helyi óra" impulzussorozatával mintavételezheti a vett jel minden bitjét. Adásszünetben a STOP-jelnek megfelelő állapot van a vonalon (2. ábra).

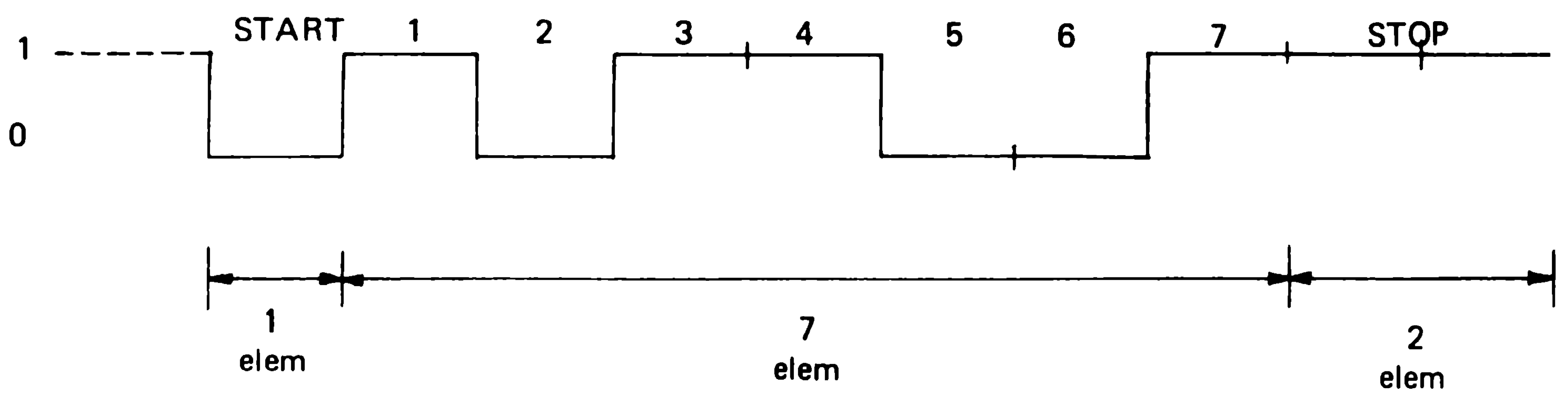
Ez a módszer a távirótechnikából származik, a távadatfeldolgozás adatátviteli rendszerében is elsősorban a *kissebességü, dialógus jellegü* üzenetváltást végző adatállomások - pl. Teletype stb. - átviteli módja. Mivel "véletlenszerüen" beérkező karakterek továbbitására kitünően alkalmas, nagyon jól megfelel interaktiv ember-gép kapcsolat esetében. A *közepes sebességi tartomány alsó felében* - körülbelül 4800 bit/sec sebességig - folyamatosan dolgozó *puffereletlen terminálok* - például lyukszalagos berendezések - szintén aszinkron átvitellel dolgoznak. Folyamatos üzemnél a STOP-jel nem nyulik a szükséges minimumnál (1,42-szeres vagy 1,5-szörös vagy 2-szeres elemi jelhosszuság) hosszabbra, időtartamát az adóállomás automatikusan beállitja. A 2. ábrán egy öt-, és egy nyolc-bites karakter START-STOP átviteli strukturája látható.

Az aszinkron átvitelnél rendszerint - az alkalmazott kódtól függően - karakterenként 1-1 paritásbitet is átvisznek. Ez azonban csak alacsony foku hibavédelmet biztosit. Nincs akadálya az ugynevezett blokkellenőrzésnek (ld. 5. fejezet) sem, de csak nagyon ritkán alkalmazzák.

A 2. ábrán is láthatjuk, hogy az átvitt információnak alig több, mint kétharmada a hasznos adat, a vonal kihasználása nem kedvező,

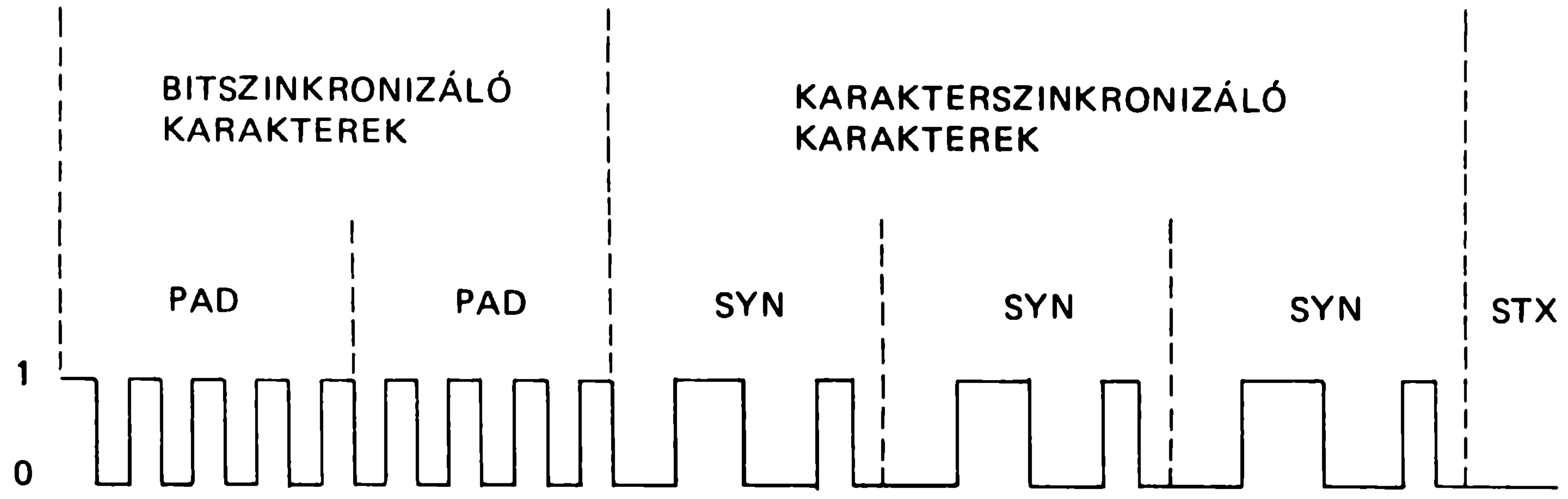

START
1
2
3
4
5
STOP
1
0
1
elem
5
elem
1,5
elem

Az „F" betü CCITT Nr. 2. kódja a vonalon


START
1
2
3
4
5
6
7
STOP
1
0
1
elem
7
elem
2
elem

Az „M" betű CCITT Nr. 5. kódja a vonalon

*2. ábra*
*Karakterstruktura, aszinkron átvitelnél*


BITSZINKRONIZÁLÓ
KARAKTEREK
KARAKTERSZINKRONIZÁLÓ
KARAKTEREK
PAD
PAD
SYN
SYN
SYN
STX
1
0

A „PAD" SPECIÁLIS KARAKTER (NINCS A KÓDTÁBLÁZATBAN), „SYN" és „STX" a 6, 7 és 8-as táblázatokban található.

*3. ábra*
*Szinkronjelek az üzenet elején*

Ráadásul egy karaktersorozatból álló üzenetben a szövegen kivül vezérlő karakterek is találhatók, amelyek szintén csökkentik az időegység alatt átvihető hasznos adatmennyiséget.

Ezekért a hátrányokért az kárpótol bennünket, hogy az aszinkron átvitel adatállomási berendezései lényegesen egyszerübbek és olcsóbbak a szinkron átvitel berendezéseinél, ezért azokban a rövid üzenetekkel dolgozó rendszerekben, ahol az alacsony átviteli sebesség is megfelel, az aszinkron átvitel alkalmazása gazdaságos.

b) *Szinkron átvitel*

Szinkron átvitel esetén a néhány karaktertől többszáz karakterig terjedő méretü adatblokkokat *folyamatos jelsorozat* formájában visszük át. A hasznos adatblokk az üzenet szöveg-részeként jelenik meg az átvitel során (ld. 6. fejezet). Az üzenet átvitelének teljes idejében az adó- és a vevő adatállomás egymással szinkronban müködik. Az üzenet összes bitjei egymással azonos elemi jelidők alatt kerülnek átvitelre.

Az adatblokk hossza lehet változó - ilyenkor általában az adathordozó, például mágnesszalag, változó fizikai rekord hosszuságához igazodik -, vagy lehet állandó, például egy közbenső puffertárolóban akkumulált adatok átvitelénél.

Gyorsitja az effektiv adatátvitelt, hogy a karaktereket nem választják el START és STOP-bitek egymástól.

Minél hosszabbak a blokkok, relative annál rövidebb időt igényel a szinkronizálás és annál kedvezőbb a szöveg aránya a vezérlő karakterekhez, tehát az effektiv adatátvitel annál gyorsabb. Természetesen hosszu blokkoknál nagyobb az átviteli és igy a nem javitható hibák valószinüsége is, nehezebb a szinkronizmus tartása is, ezért az adatblokkok méreténél ésszerü kompromisszumra kell törekedni.

Az adatállomások szinkronizálása többféle módon történhet.

A leggyakoribb megoldás, hogy az adatátvitelt néhány karakterből álló szinkronjel előzi meg, amely a vevőoldali oszcillátort az adóoldali oszcillátorhoz szinkronizálja.

Az oszcillátorok pontossága akkora, hogy a szinkronizmust a teljes üzenet átvitelének idejére nagy biztonsággal tartani tudják.

Az oszcillátor lehet a modemben vagy az ezt vezérlő logikában is.

Más rendszerekben a szinkronizmus fenntartásáról ugy gondoskodnak, hogy előre meghatározott, például 1 másodperces időközönként szinkronjelsorozatot illesztenek az adatok közé.

A szinkronjelsorozat mindkét fenti esetben egy máskor elő nem forduló jelkombináció, amit az adóoldal felismer, és igy oszcillátorát hozzáigazithatja.

Egy további, ritkábban használt megoldás, hogy az adatokat hordozó csatornaszimbólumok (0, 1) mellett egy harmadik, periódikusan közbeiktatott csatornaállapot is van, amely a szinkronizmust biztositja. Ebben az esetben nincs szükség külön szinkronizáló karakterekre, viszont a csatorna sávszélessége nem használható ki.

Viszonylag ritkán, *párhuzamos szinkron átvitelt* is alkalmaznak, ebben az esetben rendszerint egy külön parallel csatornán haladnak a szinkronjelek.

Az adatállomások közötti szinkronizmust adásszünetben is fenntartják.

A félduplex vonal két végén elhelyezett adatállomások a szinkronizmust ugy biztositják, hogy felváltva adnak, egymással szinkronjel-párbeszédet folytatnak (handshaking mode).

Egy szinkronjelsorozat végén a vezérlést átadják a másik adatállomásnak.

Duplex vonalakon a két adatállomás egyszerre küldi szinkronjeleit egymásnak, miközben felváltva birtokolják a vezérlést.

A szinkron átviteli rendszerek megbizhatósága, jel-zaj viszonya, sebessége nagyobb, mint az aszinkron rendszereké.

*Pufferelt terminálok* - display, pufferelt lyukkártyás berendezések, sornyomtatók - és *számitógép, valamint számitógépek közötti közvetlen adatcsere* nagy adatmennyiségének mozgatásához *közép- és nagysebességü szinkron adatátvitel* használata gazdaságos. Ez annak ellenére igaz, hogy a szinkron átvitel adatvégállomási berendezései sokkal bonyolultabbak és költségesebbek, mint az aszinkron átvitel berendezései, és hogy szinkronizálási hiba esetén az egész üzenetet meg kell ismételni.

Szinkron átvitelnél rendszerint hatékony hibavédelmet használnak, például két dimenziós paritásellenőrzéssel vagy ciklikus kódolással ellenőrző karakter(eke)t rendelnek az adatblokkhoz (ld. 5. fejezet).

A szinkronizálás problémáját általánosan ugy tekinthetjük, hogy tulajdonképpen bit, karakter- és üzenet-szinkronizálásra is szükség van. A *bit-szinkronizálás* azt biztositja, hogy ne a bithatáron lezajló tranziens ideje alatt, hanem a bit-idő közepének "környékén" vegyünk mintát a beérkezett jelből.

A *karakter-szinkronizálás* mondja meg, hogy a vett karakter vagy byte melyik bitje hányadik a sorban.

Az *üzenet-szinkronizálás* révén viszont a vezérlő- és szövegkaraktereket tudjuk szétválasztani, azonositani.

Ilyen értelemben az aszinkron átvitel is "üzenetre szinkronizált".

### c) *Impulzus-folyam átvitel*

Az impulzus-folyam átvitelt csak ritkán, kis földrajzi kiterjedésü, üzemen, intézményen belüli, telephelyi (in-plant) rendszerekben használják.

Viszonylag nagy sebesség érhető el kevés átviteli vonal és nem tul költséges, de speciális kiképzésü adatállomások segitségével. Az egyszerü, sodrott huzalpárból vagy koaxiális kábelből álló vonalat hurok formájában körbevezetjük a vezérállomás adatátviteli vezérlő egységétől kezdve az összes felfüzött terminál-vezérlő egységeken.

Minden egységben - vagy a vonalszakaszok hosszától függően még sürübben - jelregeneráló, ismétlő áramkörök foglalnak helyet, igy nagy, mintegy 500 000 bit/s sebesség érhető el.

A folytonos impulzussorozat rövid, egyetlen karakterből és az őt irányitó jelcsoportot tartalmazó fix hosszuságu *adathordozó blokkokból* és *szinkronjel blokkokból* áll. A hurokra felfüzött különböző adatállomások véletlenszerüen, aszinkron módon helyezhetnek el egy-egy karaktert az éppen üres adathordozó blokkba. Az adatblokkból az irányitó jelcsoporttal megcimzett adatállomás kiválasztja a neki szóló adat-karaktert. A vezérlő byte segitségével az adatbyte jelentése módositható.

## 3.4. *Analóg és digitális adatátvitel*

*Analóg és digitális vonalakat* egyaránt felhasználhatunk adatátvitelre.

a) *Analóg adatvonalak*

*Analóg vivőfrekvenciás átvitelről* akkor beszélünk, ha az eredeti digitális jelet *frekvencia transzlációnak, modulációnak* vetjük alá. Igy az egyenáramu jelkomponens, illetve az alacsony frekvenciák közvetlen átvitelére nem alkalmas *analóg* csatornákon is áthaladhat a jel. A hasznos jel egy "$f_V$" vivőfrekvencia környezetében jelenik meg a moduláció fajtájától függő sávszélességtartományban. A vivőfrekvenciás átvitelt analóg jellegü, *távbeszélő tipusu hálózatok* frekvencia-osztásos rendszerében alkalmazzák.

A hálózatban a vonalak mentén fellépő csillapitások ellensulyozására analóg jelerősités történik, a kapcsolási pontokon egyszerü vonalkapcsolást alkalmaznak. Az adatállomások modemjeik segitségével csatlakoznak a vonalra.

A vivőfrekvenciás rendszerek három alapvető modulációs eljárást alkalmaznak:

1) *Amplitudó-moduláció* (AM)

Az amplitudó modulációt - elsősorban alacsony jel-zaj viszonya miatt - adatátvitelre viszonylag ritkán, főleg rádió-táviratozás jellegü adatátvitelnél, használják. Az eredeti 0-$f_1$ jelspektrum felkerül az $f_V \pm f_1$ frekvenciasávba.

2) *Frekcencia moduláció* (FM)

A frekvencia moduláció előnye az amplitudó-modulációval szemben, hogy az adatjel minden állapotában van kimenőjele, igy nagyobb átlagteljesitmény, nagyobb jel-zaj viszony érhető el vele. Hátránya a nagy relativ sávszélesség. A gyakorlatban közepes sebességü átvitelig ez a modulációs eljárás a legelterjedtebb.

Az egyes csatornaállapotoknak egy-egy diszkrét frekvencia felel meg. (FSK = frequency shift keying).

3) *Fázis-moduláció* (PM)

A fázis-moduláció relativ sávszélessége és jel-zaj viszonya is kedvező, gyakran alkalmazzák a közepes-, legelterjedtebben a nagysebességü analóg rendszerekben. A logikai bemenő szinteknek egy referencia-fázistól adott értékkel eltolt fázisu jelek felelnek meg.

Nemcsak az amplitudó-, hanem a frekvencia- és fázis-modulációt is használják kettőnél *többszintü* modulációs eljárásokban. A *modulációs szintek száma* tulajdonképpen az *elemi vonalállapotok* száma, ezért az "n" szintü modulációval ("n" tipikus értéke: 2, 4, 8, 16) a bináris csatornához képest $\log_2 n$-szeres információmennyiség vihető át időegység alatt. Leggyakoribb a négyszintes moduláció, amellyel egy adott vonalon kétszeres adatátviteli sebesség érhető el.

Az adatátviteli tulajdonságok javithatók a kombinált modulációs eljárások - például AM/PM - alkalmazásával.

Igen nagy sebességü adatátviteli rendszerekben a fenti modulációs eljárások kombinációinak többszintes változataival is találkozhatunk.

b) *Digitális adatvonalak*

Ha a bináris (digitális) információt frekvenciatranszláció nélkül adjuk a vonalra, akkor *alapsávi digitális átvitelről* beszélünk. Ilyen digitális adatvonal az *egyenáramu átvitellel* dolgozó hagyományos *távíró-* és *telex-vonal*. Alacsony sebességü adatátvitel céljára ezek megfelelnek. A nagysebességü és nagytávolságu digitális átvitelt mintavételezésen alapuló kódmodulációs eljárással realizálják.

1) *Alapsávi digitális átvitel*

Egy egyszerü sodrott huzalpáron néhány kilométer távolságra is elfogadható minőségü egyenáramu átvitelt érhetünk el 300 bit/s sebességig. Ennél nagyobb sebességgel a huzalpárt csak rövid távolságokra használhatjuk a vezetékek ohmos vesztesége, kapacitása és induktivitása miatt.

Nagy sebességü egyenáramu átvitel érhető el koaxiális kábelek használatával, illetve a vonal mentén periódikusan elhelyezkedő jelregenráció, ismétlő-áramkörök alkalmazásával.

A közvetlen egyenáramu átvitel igen gazdaságos megoldás például in-plant időosztásos rendszerekben, sok alacsony sebességü adatállomás esetén, mivel a vonalcsatlakozás egyszerü, nagyszámu drága modem takaritható meg.

Sokszor az eredetileg analóg átvitel céljára épült - például hangfrekvenciás - vonalakat is használják alapsávi digitális átvitelre. Ebben az esetben a vonalcsatlakozás valamivel bonyolultabb feladat. Ilyen speciális célra épült vonalcsatlakozó berendezés a SIEMENS GDN, amely telefonvonalra dolgozik.

Egyik fajtája például rövid szakaszokon 4800 bit/s sebességü alapsávi átvitelt tesz lehetővé. Az ilyen "alapsávi modem" egyszerübb felépitésü, mint a vivőfrekvenciás átvitel céljára épült modemek.

### 2) *Pulzus-kód moduláció*

*Mintavételes digitális átviteli* technika a *pulzus-kód moduláció* (PCM). Ez csupán egyik, de az adatátvitel számára igéretes és nagy jelentőségü fajtája a lehetséges - mintavételezésen alapuló - impulzus modulációs eljárásoknak.

A mintavételezés alapja az, hogy az $f_M$ maximális frekvenciáju jelből másodpercenként vett $f_S \geq 2\ f_M$ számu minta elegendő a jel rekonstruálásához. Például az $f_M = 400$ Hertz sávszélességü beszédcsatorna átviteléhez $f_S = 800$ minta szükséges másodpercenként. Ha az egyes minták "m" féle diszkrét értéket vehetnek fel (kvantálás), akkor a minták bináris megjelenitéséhez (kódolás) $d \geq \log_2 m$ bitre van szükség. A minták kódjai közötti szünetekben további jelekből vett minták bináris kódcsoportjai helyezhetők el. Igy "n" különböző, $f_M$ sávszélességü jel átviteléhez, valamennyi jelből vett összes mintát "d" bites kódcsoportokkal kódolva $f_{PCM} \geq 2 \cdot n \cdot d \cdot f_M$ PCM sávszélesség szükséges.

Ezzel tulajdonképpen az "n" féle jelet idő-multiplex módon viszszük át. A nagy sávszélességért bőségesen kárpótol bennünket a periódikusan elhelyezett jel-regeneráló, ismétlő-áramkörök segitségével biztositott nagy jel-zaj viszonyu, vagy megbizhatóságu átvitel, valamint az adatállomások költségeinek csökkenése a frekvencia-multiplex rendszerekhez képest.

(Az idő- és frekvencia-multiplex átvitelre a 3.5 pontban még visszatérünk.)

A PCM rendszerekben együtt haladnak a vonalon az analóg jelekből - telefon, televizió stb. - mintavételezéssel, kvantálással és kódolással a fentiek szerint nyert, és az eredetileg is digitálisan megjelenő jelek, valamennyien digitális formában. A PCM rendszereket a technológiai fejlődés mai tendenciái egyre gazdaságosabbá teszik, igy várható, hogy az ilyen tisztán digitális rendszerek mind az analóg jelek átvitelében, mind az adatátvitelben egyre nagyobb szerephez jutnak az elkövetkező időkben.

A különböző sebességü alapsávi digitális vonalak PCM vonalakba, a PCM vonalak nagyobb sebességü PCM hálózatágakba foghatók össze és igy *hierarchikus digitális adatátviteli hálózatot* képezhetnek. Ilyen rendszer az Amerikai Egyesült Államokban az ATT-Bell által tervezett országos digitális hálózat, melynek egyes csatornái, - mint például a "PICTUREPHONE" képtelefon szolgálat - már müködnek. /Irod. 5./

Igen nagy jelentőségü az a nemzetközi törekvés, hogy *Egységes Digitális Távközlési Hálózatot* hozzanak létre. Igéretes lépés az egységesités irányába az Európában elsőként a Német Szövetségi Köztársaságban létrehozott DATEX hálózat, amely kis-, közepes- és nagysebességü változatban is kiépült, illetve fejlesztés alatt áll.

DATEX-jellegü digitális adatátviteli szolgáltatás megvalósitása több országban is folyamatban van, 1971 eleje óta hazánkban is létezik a DATEX-szolgáltatás. Az Egységes Digitális Távközlési Hálózatba nagyon jól illeszkedhetnének a már meglévő PCM rendszerek is. /Irod. 4., 6., 7./

### *3.5. Az adatvonalak kihasználásának fokozása*

A nagykiterjedésü, sok adatvonalból álló adatátviteli hálózatra épült távadatfeldolgozó rendszerekben az adatvonalak fenntartási, illetve bérleti dijából származó ***kommunikációs költséghányad*** az összköltséghez képest is ***jelentős***, sőt ***néha domináló*** lehet. Kézenfekvő tehát az a törekvés, hogy az adatátvitel ***költségeit*** az adatvonalak ***kihasználásának fokozásával*** és az adatátviteli ***hálózat struktúrájának optimalizálásával csökkentsük.***

Az integrálódási folyamat előrehaladásával egyre jobban előtérbe kerülnek ezek a problémák is. A megoldás mindinkább a ***számitógépnek*** az adatátvitel területére történő ***behatolásától várható***, tehát a távközlési és számitástechnikai szakemberek intenziv együttmüködését feltételezi. /Irod. 8., 9., 10./

Az adatvonalak fokozott kihasználásának lehetősége két tényezőn alapul:

a) Az ***adatvonalak igénybevételi költségei*** - adott távolságu átvitel esetén - a sávszélességgel ***degresszíven növekednek.***

Napjainkban ez elsősorban az alacsony-, és középsebességü vonalak költségarányára igaz. Általában kisebb - de várhatóan egyre növekvő - mértékben teljesül azonban középsebességtől felfelé is.

Tipikus értékként: a bérelt távbeszélő vonalak költsége átlagosan kétszerese a bérelt táviróvonalakénak, holott a távbeszélő vonalakon elérhető adatátviteli sebesség legalább husszoros, de rendszerint még több.

b) A ***terminálok többsége*** a "vonalon töltött" ***összidőnek*** csak igen ***kis*** - átlagosan tiz körüli - ***százalékában folytat tényleges kommunikációt.*** Ez elsősorban szintén a főleg az alacsony sebességtartományba eső interaktiv ember-gép kommunikációra igaz, bár kisebb mértékben minden más esetben is fennáll.

Ezeket a tényezőket az adatvonalak tér-, frekvencia- és időosztásos üzemeltetésével, ***multiplexerek*** és ***koncentrátorok*** alkalmazásával lehet a költségek csökkentésének irányában kihasználni. Az időegység alatt átvitt információra eső költséghányad a vonalak kihasználásának fokozása miatt, a járulékos berendezések ellenére, csökkenhet.

Tulajdonképpen az adatvonalak kihasználásának fokozását szolgálja a felfüzéses hálózatok és a kapcsolt hálózatok alkalmazása is. Ezek általános jellemzőit a következő fejezetben vizsgáljuk meg. Most csupán a multiplexelés és koncentrálás fajtáit tekintjük át röviden, ahol - szemben a fentiekkel - a távadatfeldolgozó rendszer felhasználójának általában nem kell törődnie az adatátviteli hálózatban lejátszódó folyamatokkal.

Mielőtt a különböző technikák lényegét megvizsgálnánk, határozzuk meg a multiplexelés és koncentrálás ismérveit, mivel e kifejezések használata néha megtévesztő.

A *multiplexelés* olyan eljárás, amelynek során egy adatvonalat *előre meghatározott*, rögzitett *módszer szerint osztunk fel elemi adatcsatornákra*. A dolog *statikus* jellege miatt a multiplexer teljes input és teljes output *kapacitása egymással megegyezik*. Minden elemi bemenő csatornához elemi kimenő csatorna is tartozik, ezért a multiplexelés *nem okozhat csatorna-foglaltságot*. Az igények kielégitése várakoztatás nélkül történik, a multiplexelt adatvonal-szakasz transzparens, az elemi adatcsatornák folytonosak vagy annak tűnnek, a felhasználók nem érzik a multiplexelést.

Ezzel szemben a *koncentrálás* olyan eljárás, amelynél adott számu elemi bemenő adatcsatorna *dinamikusan és igény alapon osztozkodik* kevesebb számu kimenő adatcsatornán. A dinamikus hozzárendelés éppen azért szükséges, mert *a teljes output kapacitás kisebb, mint a teljes input kapacitás*. A koncentrátor arra törekszik, hogy az adatátviteli forgalmat egyenletesebbé tegye, ilyen *forgalomkiegyenlitő hatása* a multiplexernek nincs. A koncentrátorok kihasználják az *üzenetek statisztikus jellegét és sorbaállithatóságát*. Mivel a ki- és bemenő adatcsatornák összerendelése igény alapon történik, a *koncentrátor* a felmerülő átviteli igények torlódása esetén *jelezhet foglalt állapotot* is, tehát az igények kielégitése néha csak *várakoztatással* történhet meg.

A multiplexelés és a koncentrálás elvét a távközléstechnika ismerte és alkalmazta már régebben is, ezért lényegében csak a módszerek adaptálására és továbbfejlesztésére volt szükség.

Az alkalmazott eljárások a következők:

1) *Multiplexelés frekvenciaosztással* (FDM)
(Frequency-division multiplexing)

Frekvencia-multiplex üzemmódban elsősorban a távbeszélő hálózatok vivőfrekvenciás rendszereinek szélessávu fővonalait használják. A széles frekvenciasávban időben is egyszerre haladnak a különböző vivőfrekvenciákra ültetett jelek.

Az egyes elemi sávszélesség-tartományok között elválasztó frekvencia-résre van szükség, mivel a különböző jeleket szétválasztó szürők meredeksége véges. A frekvencia-rések jelentősen csökkentik a fővonal sávszélességének kihasználhatóságát. Ráadásul az éppen nem dolgozó berendezésekhez rendelt frekvenciasávok is kihasználatlanok.

A sávszélességet általában az ekvivalens 4KHertz-es beszédcsatornák számával adják meg. Nemcsak a fővonalat, hanem a normál távbeszélő vonalakat is feloszthatjuk. A beszédcsatornát leggyakrabban 12, ritkábban 24 azonos sávszélességü *távíró-csatornára osztják.*

Tizenkét beszédcsatornát *távbeszélő csoportba* fognak össze, amely nemzetközileg a 60-108 KHertz-es frekvencia-tartományban fekszik. Öt távbeszélő csoport egy *távbeszélő főcsoportot* alkot, ez 240 KHz sávszélességü. Tiz főcsoportból jön létre egy 2,4 MHz sávszélességü *bázis-mestercsoport.*

Ezeket is 3-as vagy 6-os *mestercsoportokba* lehet foglalni.

A frekvencia-multiplex berendezéseknél egy adott szélessávu rendszeren minden további alcsatorna kiépitése tetemes mennyiségü járulékos beruházást (szürőrendszert) igényel, a bővités meglehetősen költséges.

A frekvencia-osztás előnye, hogy az egyes alcsatornák egymástól földrajzilag eltolva kezdődhetnek és végződhetnek, a csoportba fogott jeleknek nem szükséges kis körzetben elhelyezkedő adatállomásokhoz tartozni.

Természetesen a multiplexelt vonal minden "megcsapolásánál" külön multiplexer szükséges. A frekvencia-multiplexer általában a modem funkcióit is ellátja, ezért a külön modemek megtakarithatók.

2) *Multiplexelés szinkron időosztással* (STDM)
(Synchronous time-division multiplexing)

Az *idő-multiplex* eljárásról tulajdonképpen már beszéltünk a pulzus-kód-moduláció kapcsán.

Az idő-multiplex berendezések a nagyobb sávszélességü adatvonalat időben osztják fel több, elemi csatornára.

Minden elemi csatorna egy időszeletet kap. A fővonal két végén elhelyezkedő vonali multiplexerek előre meghatározott ütemben, periódikusan, egymással szinkronban müködve összekapcsolják egy-egy rövid időre - néha egyetlen bit, legtöbbször egyetlen karakter vagy byte, esetleg néhány byte idejére - az összetartozó be-, illetve kifutó vonalakat.

Idő-osztásnál a fővonal sávszélessége jobban kihasználható, mint frekvencia-osztásnál.

Néha a nagysebességü fővonal közvetlenül a feldolgozó számitógéphez (illetve annak adatátviteli vezérlő egységéhez) csatlakozik és a demultiplexelés feladatát a számitógép látja el.

Elvileg a szinkron és az aszinkron átvitelnél egyaránt használhatunk bit-multiplex és karakter-multiplex eljárást is, sőt szinkron és aszinkron vonalakat együtt is multiplexelhetünk.

Az aszinkron átvitellel dolgozó adatvégállomások jeleit azonban általában karakterenként multiplexelik. A multiplexer az átvitel során elhagyhatja, majd a kimenetnél, demultiplexelésnél rekonstruálhatja a Start-Stop biteket, ezzel időt nyer.

A szinkron átvitel jeleit rendszerint bit-multiplex eljárással multiplexelik.

Bármilyen tipusu az átvitel és bármekkora is a multiplexelt információegység, szükség van arra, hogy a vonal két végén elhelyezett multiplexerek szinrkonizmusát biztositó periódikus jeleket is elhelyezzünk az információegységek között. Ezek a szinkronjelek kissé csökkentik a fővonal kihasználhatóságát.

Az időosztásos elven müködő multiplexerek kaszkádba kapcsolása, vagyis leágazások készitése a fővonalon, szinkronizálási nehézségeket okoz, ezért az STDM eljárás használata akkor előnyös, ha a távolvégi, adatvégállomási multiplexer közelében egy csoportban több terminál helyezkedik el.

A *távolvégi multiplexer* egyben betöltheti a terminál egységek közös vezérlő egységének szerepét is.

A szinkron idő-multiplex eljárás hátránya, hogy az éppen nem dolgozó terminálokhoz is elemi adatcsatornát rendel s igy - bár a fővonal rendszerint dolgozik - az elemi csatornák egy része csaknem mindig kihasználatlan.

Léteznek olyan *aszinkron idő-osztásos multiplex* berendezések is, amelyek az időszeletek, tehát az elemi csatornák és a terminálok összerendelését bizonyos korlátok között az igényeknek megfelelően képesek megváltoztatni. Ezek a berendezések tulajdonképpen a multiplexelés és a koncentrálás előnyeit igyekeznek egyesiteni bonyolult hardware vezérlő egység segitségével. Az ilyen berendezések egyelőre még nem terjedtek el.

### 3) *Vonalkoncentrálás* (line concentration)

A vonalkoncentrátorok speciális vonalkapcsoló központok. Különlegességük abban áll, hogy nem azonos vonaltipusok közötti tetszőleges összekapcsolások létrehozására valók, mint a vonalkapcsoló központok általában.

A vonalkoncentrátor célja az, hogy az adatátviteli hálózat magasabb hierarchiáju ágainak jobb kihasználása érdekében *nagyszámu alacsony hierarchiáju* vonalat *kisebb számu magasabb hierarchiáju* vonalhoz kapcsoljon az éppen felmerülő igényeknek megfelelően.

A vonalkoncentrátorokat elsősorban a vonalkapcsolásos hálózatok előfizetői szintjén alkalmazzák, ezért a bővebb oldalt előfizetői, - a szükebbet törzs-oldalnak is nevezik. Az előfizetői, - és törzsvonalak azonos sebességüek.

A vonalkoncentrátor nem képes az azonos vonaltipusok közötti összeköttetések létrehozására, csupán előfizetői vonal-törzs-vonal kapcsolatot tud felépiteni.

Feladatát önállótlanul látja el, ez azt jelenti, hogy vezérlő jeleket sem feldolgozni, sem generálni nem képes, ezeket csupán a felette álló vonalkapcsoló központ vezérlésének hatására továbbitja. Egyedül a hozzákapcsolt törzsvonalak foglaltságát jelzi önállóan az előfizetői adatállomások felé.

Az adatvégállomási szintről néha vonalkoncentrátorok láncolatán juthatunk el a számitógépes adatállomáshoz; az ilyen adatátviteli hálózatot *vonalkoncentrátoros hálózatnak* nevezzük.

A vonalkoncentrátorokat elterjedten alkalmazzák a hosszu törzsvonalak jobb kihasználása céljából a hagyományos vonalkapcsolt hálózatokban.

4) *Adatkoncentrálás* (data concentration)

Az adatkoncentrálás tipikusan "intelligens vezérlést" igénylő feladat. Tulajdonképpen ugy is felfogható, mint egy olyan aszinkron időmultiplex eljárás, ahol a multiplexelt információegységek üzenet méretüek. Az adatkoncentrátor az *üzenet-kapcsolás elvén* müködő berendezés. Mig azonban az üzenetkapcsoló központ tipikusan egyenrangu vonalak közötti kapcsolatokat teremt átmeneti üzenettárolás segitségével, addig az adatkoncentrátor tetszőleges vonalakat koncentrál egyetlen (vagy kisszámu) nagysebességü fővonalba, amelynek kapacitása azonban kisebb, mint a koncentrált vonalak összkapacitása. Az adatkoncentrátor a viszonylag lassu adatvégállomásoktól származó üzeneteket összegyüjti, majd a fővonal maximális sebességével továbbitja a számitógépes adatállomás felé. A feldolgozó központból sorosan, nagy sebességgel érkező üzeneteket fogadja, átmenetileg tárolja és szétosztja, rendeltetési helyeik szerint.

Rendszerint a kis sebességü aszinkron mellékvonalakat nagy sebességü szinkron fővonalba koncentráljuk, de a mellékvonalakon is dolgozhatnak szinkron átvitelt használó terminálok. *A forgalomkiegyenlitő hatás* mindenekelőtt az adatkoncentrátor átmeneti tárolásának köszönhető.

Az adatkoncentrátorba bejutó üzenetek hossza általában "tetszőleges" lehet.

Minden üzenetet vezérlő és irányitó jelek juttatnak el a rendeltetési helyre, a termináloknál ülő felhasználóknak azonban az adatkoncentrátor közbeiktatása nem jelent többletmunkát.

A röviden vázolt fő feladat mellett az adatkoncentrátorok számos járulékos szolgáltatást is tehetnek, ezekre a 10. fejezetben még visszatérünk.

# 4.

# Adatátviteli hálózatok jellemzői

A ma müködő távadatfeldolgozó rendszerek tulnyomó többsége adatátviteli hálózatnak a már korábban meglévő távközlési hálózatot használja, de léteznek kimondottan a távadatfeldolgozó rendszerekhez létrehozott speciális hálózatok is. A számitógép nem csupán mint feldolgozó központ kapcsolódhat az adatátviteli hálózathoz, hanem a hálózat szerves részeként, - kapcsoló-központ, multiplexer koncentrátor stb. (ld. 4.3., 10. és 11. fejezet) - is egyre szélesebb körben használjuk a hagyományos hirközlő hálózatok megfelelő hálózatelemei mellett vagy helyett, illetve azok továbbfejlesztett változataiként.

Mindenesetre a számitástechnika és a távközléstechnika integrálása világszerte nagy lendületet adott a távközlési hálózatok fejlesztésének, megbizhatóságuk növelésének is.

A későbbiek során csupán a távadatfeldolgozás szempontjából legfontosabb kérdésekkel: a hálózatok topológiájával, az összeköttetések tulajdonságaival és a hálózatok igénybevételének kérdéseivel foglalkozunk.

## 4.1. *Hálózatok topológiája*

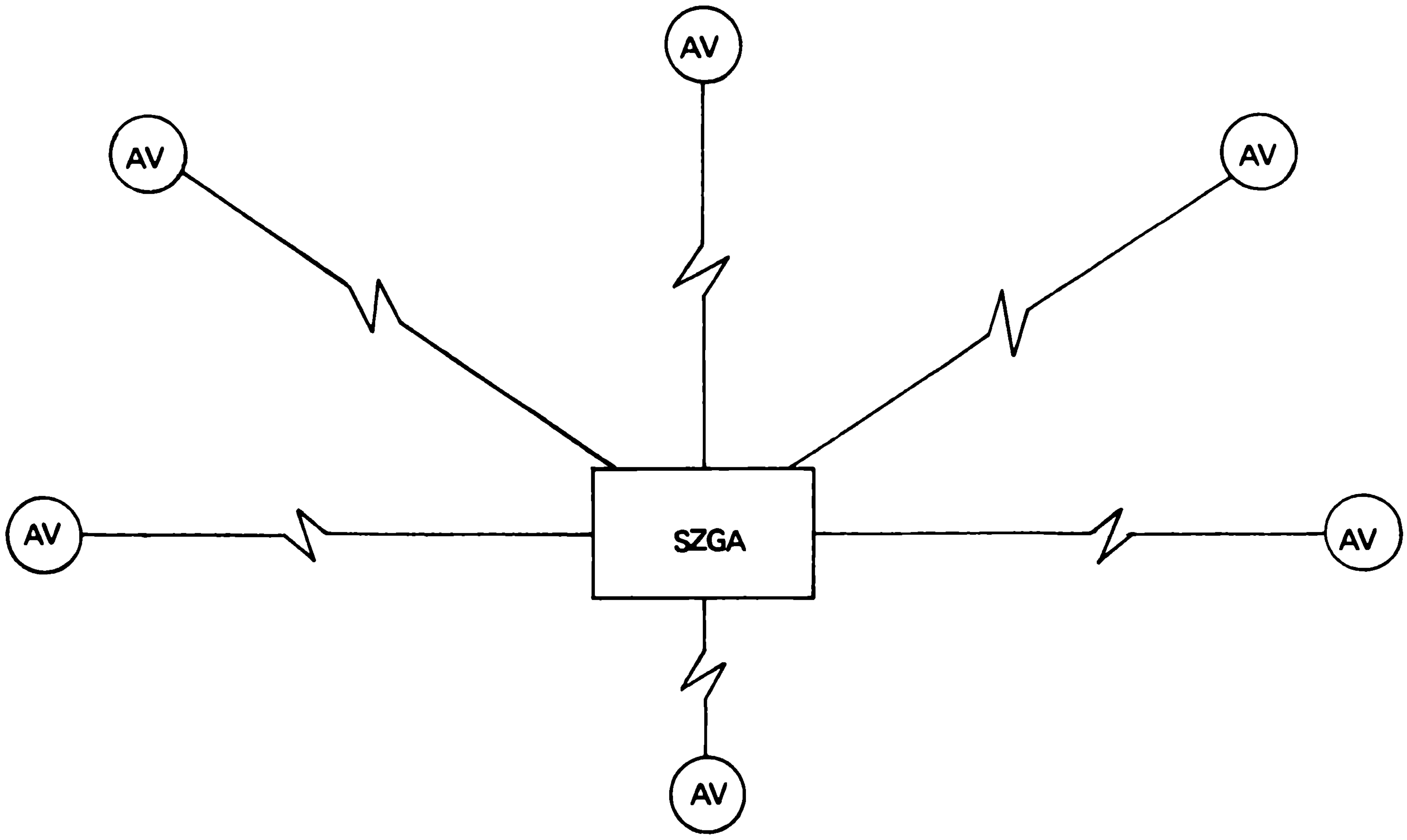
Az adatátviteli hálózatok kiépitésénél a hagyományos távközlési hálózati formák valamelyikét követik.

Az egyetlen feldolgozó központot és a köré telepitett adatvégállomásokat összekapcsoló hálózat gyakori formája a *csillaghálózat* (ld. 4. ábra).

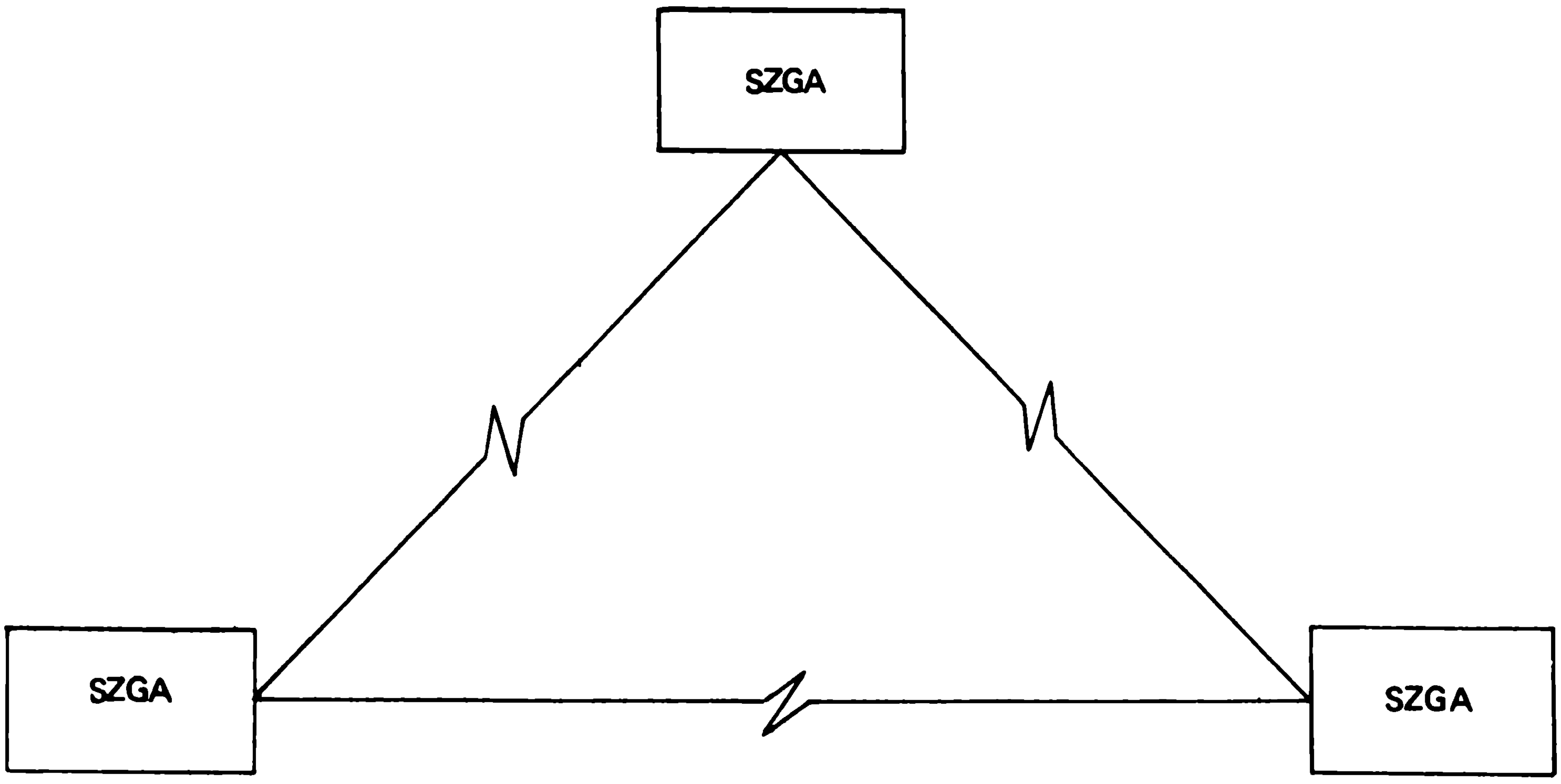
Ez a forma akkor előnyös, ha a feldolgozó központ a "geometriai középpont" környezetébe telepithető, az egyes adatvégállomások nem esnek tul közel egymáshoz a számitógéptől való távolságukhoz képest és/vagy a saját vonalukat az idő nagy részében lekötik.

Távadatfeldolgozó hálózatok kiépitésének kezdetén - egyszerü szervezése miatt - talán ez a leggyakrabban használt hálózati forma. Bonyolult rendszerekben a csillagforma több hierarchiai szinten ismétlődhet.

A csillagforma igen érzékeny a *csomópont* és a *fa-strukturáju*, csillagágakat összefogó *vonalnyalábok* meghibásodására. A hálózat megbizhatósága nagymértékben fokozható, ha *kerülőutakat* hozunk létre a csillagágak egyenrangu végpontjai között.


AV
AV
AV
AV
SZGA
AV
AV

*4. ábra*
*Csillaghálózat*


SZGA
SZGA
SZGA

**(Az egyes számítógépes adatállomásokhoz csak helyi perifériák és terminálok tartoznak.)**

*5. ábra*
*Hurkolt hálózat*

A kerülőutak, vagy más néven *harántösszeköttetések* hozzárendelése többféle módon történhet. Az adatforgalom átterelése elvégezhető manuálisan vagy automatikusan, segédfokozatok beépitésével vagy a vezérlés bővitésével.

Olyan bonyolult hálózatokban, ahol többé-kevésbé egyenrangu feldolgozó központokat kell összekapcsolni, leggyakoribb a *hurkolt*- vagy más néven -*rácsos* - hálózati forma.

Számitógépek összekapcsolása esetén előfordul, hogy *bármely két központ között közvetlen kapcsolatot létesitenek*, - ez a *teljes hurokhálózat* (ld. 5. ábra) - ami az átviteli idő és az átvitel megbizhatósága szempontjából egyaránt előnyös. Ennek persze az az ára, hogy a hálózatágak az idő nagy részében kihasználatlanok lesznek. Számitógépek összekapcsolásakor az átviteli vonalak nagysebességüek. Terminálok szintjén ezt a teljes hurokformát igen ritkán, az egyszerü hurok mentén való *felfüzést* annál gyakrabban alkalmazzák (ld. később).

A hurkolt adathálózatok egyes feldolgozó központjaihoz csaknem minden esetben kapcsolódnak további - például kisszámitógépes, vagy egyszerü terminálokból álló - adatállomások. Ezek az "alhálózatok" csillaghálózat formájában szervezettek, igy végeredményben a 6. ábra szerinti *vegyes hálózathoz* jutottunk.

## 4.2. *Összeköttetések tulajdonságai*

Az adathálózat célja az adatállomások közötti összeköttetések létrehozása. Két kiragadott adatállomás közötti hálózatágon három alapvető *üzemmódban* történhet összeköttetés:

a) *Szimplex összeköttetés*

A két adatállomás rögzitett adó-vevő relációban van egymással, egyik mindig csak adhat, másik csak vehet, tehát a hasznos információ csak egy irányban haladhat a hálózatágon.

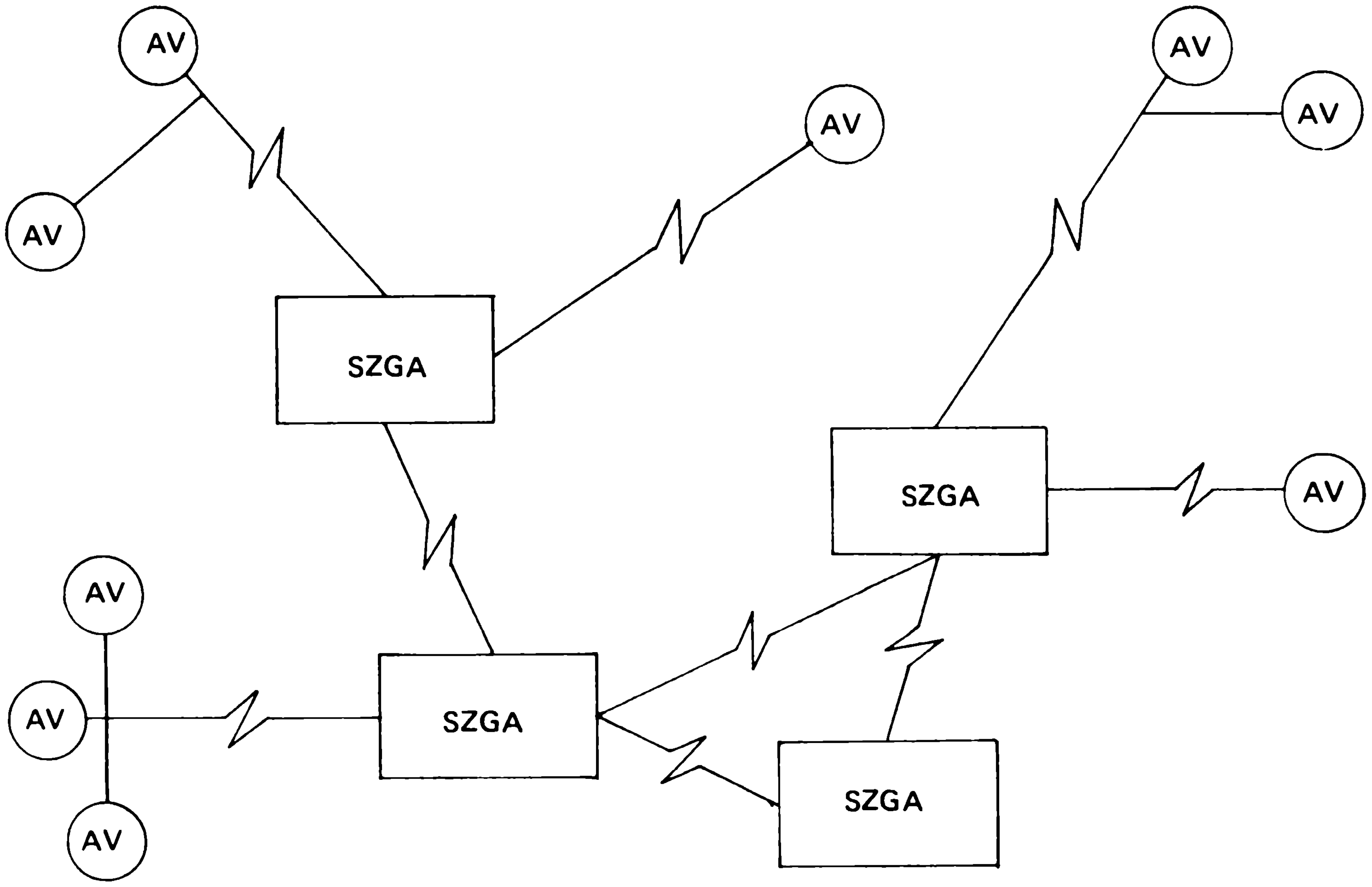
b) *Fél-duplex összeköttetés*

Mindkét adatállomás felváltva küldhet és kaphat adatokat, de egyidejüleg csak egy irányban folynak a hasznos adatok.

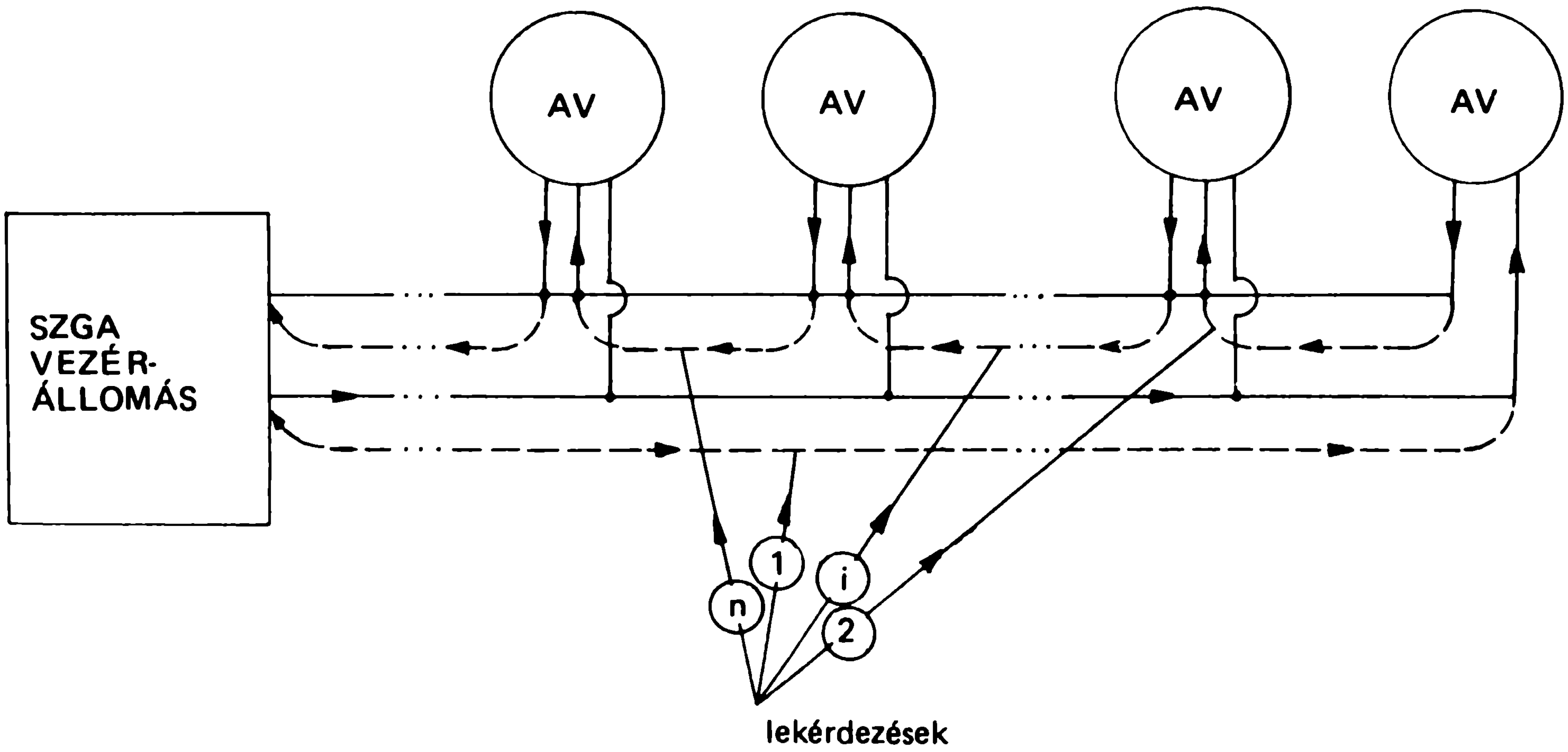
c) *Duplex összeköttetés*

Mindkét adatállomás egyidejüleg adja és veszi a hasznos adatokat, az összeköttetés mindkét irányában egyidejüleg fennáll a szövegáramlás.

Az összeköttetési üzemmódok nem azonosak a vonali üzemmódokkal, mivel nem az adatcsatorna üzemére vonatkoznak, hanem a tényleges *hasznos adatforgalomra*. A hasznos adatokon kivül az összeköttetés során vezérlő jelek is áramlanak a két adatállomás között. Ezek áramlása történhet az adatforgalommal egyidejüleg vagy vele váltakozva, az adatmozgással megegyező vagy vele ellentétes irányban. Sok esetben a vezérlő jelek külön "felügyeleti csatornán" áramlanak.


AV
AV
AV
AV
AV
SZGA
SZGA
AV
AV
AV
AV
SZGA
SZGA
AV

*6. ábra*
*Vegyes hálózat*


AV
AV
AV
AV
SZGA
VEZÉR-
ÁLLOMÁS
1
n
i
2
lekérdezések

*7. ábra*
*A lekérdezések láncolása*

Természetesen az összeköttetési üzemmódok létrehozásának nem csupán az adatvonal megfelelő tulajdonsága, hanem a müködtető adatállomásoknak az a képessége is feltétele, hogy az adott összeköttetési üzemmódban használni tudják a vonalat.

A duplex összeköttetésekhez jóval bonyolultabb adatállomási berendezések tartoznak, mint a félduplexhez, ezért főleg a magas szintü hálózatágakban találkozunk duplex összeköttetéssel. A duplex vonalakat azonban igen gyakran alkalmazzák félduplex összeköttetések lebonyolitására is. Ebben az esetben hasznos adatok áramlanak az egyik, vezérlő információ a másik irányban, majd, ha szükséges, az irányok megcserélődnek. A duplex vonal előnye a félduplexhez képest az, hogy az irányváltási idő (modem turn around time) lecsökken. Ez gyakran változó irányu, párbeszédes átvitelnél a vonal kihasználásának jelentős növekedését eredményezheti.

Mivel a duplex vonal kiépitési költsége relative olcsóbb, mint a félduplex vonalé, sok esetben érdemesebb eleve duplex vonali rendszert létesiteni és a félduplex összeköttetéseket is ezen lebonyolitani.

Az összeköttetések létrejöhetnek előre tervezett ütemezés vagy rendszertelenül beérkezett igények alapján, lehetnek közvetlenek vagy közvetettek.

Nem csak kettő, hanem egyszerre több adatállomás is összeköttetésbe léphet egymással.

*Kétoldalu összeköttetésnél* bármelyik adatállomás adhat vagy vehet adatokat az összeköttetés teljes idejében, időben váltakozva vagy időben egyszerre, az adatállomások és az adatvonal jellemzőitől függően.

Kétoldalu összeköttetésnél az adatállomások kapcsolata az adatátvitel kezdeményezése szempontjából rendszerint egyenrangu. Az összeköttetés állandóan a két oldalon elhelyezkedő adatállomások rendelkezésére áll.

*Többoldalu összeköttetésnél* akkor beszélünk, amikor "konferencia jelleggel" több adatállomás léphet kapcsolatba egymással, de a hálózat nem áll bármikor egy adott adatállomás-pár rendelkezésére. Itt különös gonddal kell ügyelni a kölcsönös érthetőségre és az átvitel biztonságára.

A többoldalu összeköttetés esetén mindig egy *vezérállomás* irányitja az összes többi, az adatátvitel vezérlése szempontjából neki alárendelt adatállomást. A vezérállomás lehet állandó jelleggel is a többi adatállomás fölé rendelve, de flexibilisebb az a megoldás, amikor több, vagy az összes adatállomás képes a többoldalu összeköttetés vezérállomása szerepének átvételére.

### *4.2.1. Közvetlen összeköttetés*

A kétoldalu összeköttetés létrehozásának legkézenfekvőbb módja a két adatállomás között fixen telepitett közvetlen, vagy más néven pont-pont összeköttetés.

A közvetlen összeköttetés nem állandóan aktiv vonalat jelent. Az összeköttetés valamelyik adatállomás kezdeményezésére, a másik adatállomás "felszólitása" utján épül fel. A tényleges adatáramlást az adatállomások szinkronizmusát és/vagy megfelelő üzemmódját biztositó vezérlő karakterek utján folytatott "párbeszéd"(handshaking) előzi meg, illetve kiséri.

Sok gyakorlati esetben nincs olyan nagy adatforgalom, hogy a közvetlen összeköttetés adta rögzitett kapcsolatot gazdaságosan tudnánk kihasználni. A közvetlen összeköttetés előnye viszont, hogy megbizható, és a vonalak és adatállomások tulajdonságai jól illeszthetők.

Egyszerü csillaghálózatban a terminálok közvetlen összeköttetésben vannak a feldolgozó számitóközponttal.

A feldolgozó számitóközpont és a terminálok valamilyen időmultiplex müködésü adatátviteli vezérlő egység utján - általában igény alapon - léphetnek kapcsolatba egymással.

### *4.2.2. Felfüzős összeköttetés*

A többoldalu összeköttetések tipikus -*többpontos* hálózati formája a *felfüzős összeköttetés*, amelyen tulajdonképpen leggyakrabban kétoldalu összekapcsolások, párbeszédek sorozata jön létre az adatállomás-párok valamilyen összerendelésével.

A felfüzős összeköttetés *vezérállomása* nemcsak a teljes hálózat adatforgalmát irányitja, hanem a többi adatállomás távvezérlését is végzi.

Az *alállomások* általában adatvégállomások, de lehetnek közöttük további felfüzött számitógépes adatállomások is.

Az ilyen többpontos hálózatokban elkerülhetetlenül előfordul, hogy egynél több adatállomás kivánja egyidejüleg használni a közös hálózatot. Egyszerre csupán egy adhatja jelét a hálózatra, a többi igénylőnek ki kell várnia, amig sorrakerül. Az igények összeegyeztetését a rendszer vezérállomása - egy számitógépes adatállomás - végzi. Ha a vezérállomás "látja", hogy a vonal szabad, jelzi a soronkövetkező adatállomásnak, hogy indulhat az adatátvitel. A jelzés hatására az átvitel maga elkezdődhet automatikusan - például egy puffertárból vagy másodlagos adathordozóról - vagy esetleg a jelzés csupán az operátort figyelmezteti a folyamat elinditására.

A felfüzős összeköttetések a következő két hálózati alaptipusba sorolhatók:

a) *Soros felfüzéses hálózat*

A soros felfüzéses hálózatban a felfüzés általában hurkot képez. A hálózat vezérállomása által egymás után kibocsátott kérdések és közlések - akárcsak a kiválasztott adatállomás üzenete - sorban áthaladnak a felfüzött adatállomásokon.

Egy párhuzamos hurkon - biztonsági okokból rendszerint független - adatáramlás folyhat az ellenkező irányban. A mindkét irányból érkező összes üzenetek igy minden állomáshoz eljuthatnak.

A megcimzett adatállomás feldolgozza a neki szóló kérdést vagy közlést és továbbitja az összes többit. Igy az összes közbenső adatállomás elvégezheti a rajta áthaladó üzenetek jelregenerálását is.

A rendszerben gyakorlatilag állandóan folyik üzenet, ezért a vonalak jól kihasználhatók. Ez a szervezés alig terjedt el, bár különösen kis kiterjedésü, nagysebességü felfüzéses adatvonalakon előnyös megoldást jelent

b) *Parallel felfüzéses hálózat*

A parallel felfüzéses hálózatokban a vonalhasználat megszervezésének két alapvető módszere van. A vezérállomásnak mindkét esetben rendelkeznie kell azzal a képességgel, hogy *szelektiven hivja* legalább egyik, de rendszerint bármelyik felfüzött adatállomást.

1) *Szervezés igény alapon* (contention)

Ennél a - viszonylag kevésbé elterjedt - módszernél az adatállomások "vetélkednek" a hálózat használatáért.

Ha az átviteli igény fellép és a hálózat szabad, akkor azonnal elkezdődik az adatátvitel. Ha a hálózat foglalt, az adatállomásnak várnia kell. A vezérállomásban felépül egy *várakozási sorrend*, amelynek kielégitése történhet a beérkezett igények sorrendjében, de történhet más szempontok alapján előre meghatározott prioritási sorrend szerint is.

Ha valamelyik adatállomás "szóhoz jut", saját adatátvitelét ugy kezdi, hogy szelektiven megcimzi a rendeltetési helyet (vagy helyeket) és megkérdezi, hogy az kész-e az üzenet fogadására.

2) *Szervezés lekérdezéssel* (polling)

A polling módszernél a vezérállomás - tekintet nélkül a tényleges igényekre - időről-időre *lekérdezi* az összes felfüzött adatállomást, hogy van-e közölni való üzenete. A lekérdezés az ugynevezett - *polling-lista* alapján történik, ez a lista határozza meg a lekérdezési sorrendet. A polling két módszere a következő:

2.1.) *Láncolt polling* (hub go-ahead polling)

A láncolt lekérdezés fix polling listáját az adatállomások *felfüzési sorrendje adja*. Ennél az eljárásnál a vezérállomás csupán a tőle legtávolabb elhelyezkedő adatállomást kérdezi le közvetlenül. A távoli adatállomás - ha nincs üzennivalója, vagy ha üzenetének végére ért - továbbitja a lekérdezést a vezérállomás irányába eső legközelebbi adatállomásra.

Ez addig folytatódik, amig a kérdés-lánc vissza nem érkezik a vezérállomáshoz (ld. 7. ábra). Ekkor a vezérállomás a vonalra adhatja saját üzeneteit, majd ujból hivja a legtávolabbi adatállomást és minden kezdődik elölről.

A vezérállomás rendelkezhet azzal a képességgel, hogy sürgős közlendője esetén a láncolást felfüggessze.

A módszer hátránya az, hogy az összes adatállomásoknak fokozott logikai képességekkel kell rendelkezniük, és az, hogy - ebben az egyszerü szervezési felépitésben - az erősen leterhelt adatállomások nem részesithetők előnyben a többiekhez képest.

Arról is kell gondoskodni, hogy a meghibásodott, vagy be nem kapcsolt adatállomások megkerülhetők legyenek.

Előnyös viszont - különösen rövid üzenetekkel dolgozó adatállomások esetén -, hogy a kevés vezérlő karakter és a terjedési késleltetés csökkenése miatt lecsökken a lekérdezési ciklusidő és megnő a vonal adatátbocsátása a névsor szerinti pollinghoz képest.

Bár e módszer használata hosszu és igy költséges adatvonalak esetében előnyös, alkalmazása csak kis mértékben terjedt el.

2.2.) *Polling névsor szerint* (roll-call polling)

*Ez a legelterjedtebben használt technika a felfüzéses hálózatok müködtetésére* (ld. 8. ábra), ezért a későbbiekben pollingon ezt az eljárást értjük.

A névsor szerinti polling esetében a polling listát a vezérállomás *tárolja.*

*A tárolt polling-lista programmal változtatható.* A legfontosabb, nagyobb adatforgalmu állomások természetesen többször is előfordulhatnak a listán, igy egyetlen lekérdezési ciklusban többször is sorrakerülhetnek.

A vezérállomás szelektiven hivhatja a tárolt polling-lista alapján a párhuzamos vonalakra kapcsolt bármelyik adatállomást. A hivójel valamennyi adatállomáshoz eljut, azt minden adatállomás kiértékeli, de csak a megcimzett adatállomás válaszol.

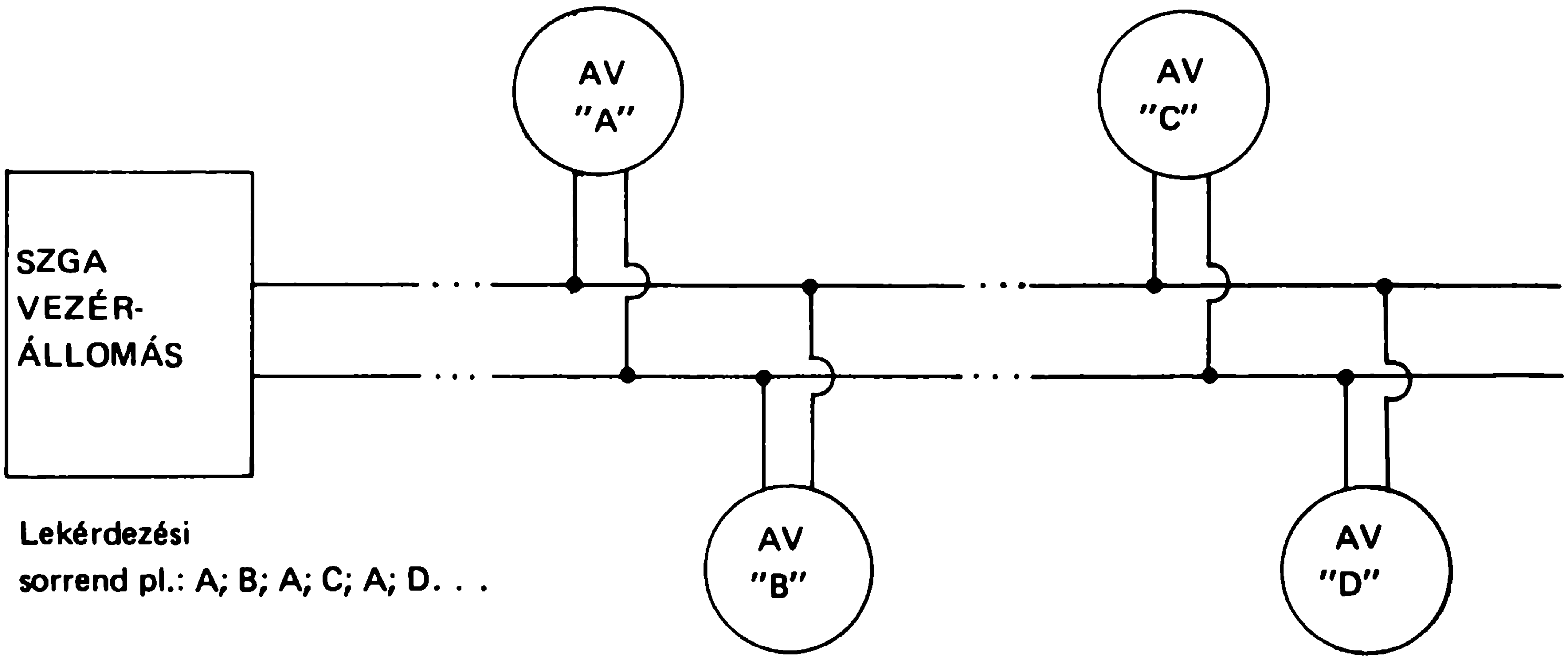
Rendszerint csak a vezérállomás képes bármelyik adatállomás szelektiv megcimzésére, az alárendelt adatállomások pedig csak a vezérállomásnak küldhetnek üzenetet. Az alállomások általában csak a vezérállomáson keresztül léphetnek kapcsolatba egymással.

Ha a vezérállomás kiván üzenetet küldeni valamelyik adatállomásnak, akkor megcimzi a rendeltetési helyet, és megkérdezi, hogy az képes-e fogadni üzenetét. Ezt az eljárást nevezik *kiválasztásnak* (selecting, vagy addressing).

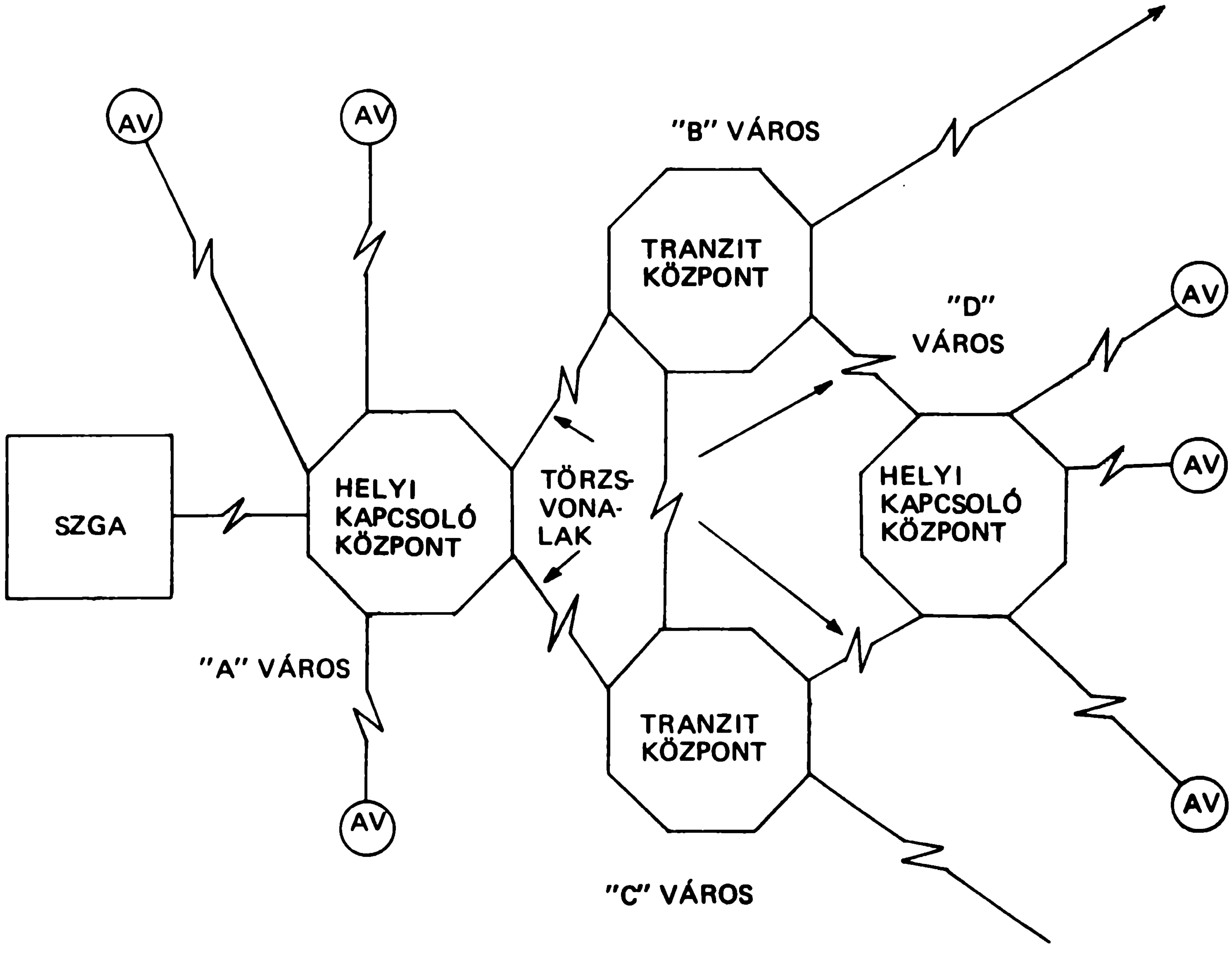
A vezérállomás néha rendelkezik azzal a képességgel, hogy egy "univerzális kóddal" egyszerre több adatállomást cimezzen meg. Ez az *együttes kiválasztás* az ugynevezett "körözvényadás", mely adatszétosztási feladatok megoldásánál előnyös.

A névsor szerinti lekérdezés speciális esete, amelynél a vezérállomás több fokozatból álló cimmel - például egy terminál vezérlő egységen át, amely több terminál egységre közös - éri el a kivánt alállomási egységet. Ez a *lépcsős lekérdezés* (drop polling).

Ha a hálózat két kiválasztott adatállomása egymással kapcsolatba lépett, akkor az átviteli procedura már ugyanugy zajlik le, mintha közvetlenül összekapcsolt, pont-pont összeköttetésben lévő adatállomásokról lenne szó. Néhány rendszerben a vezérállomásnak módjában áll még az összekapcsolás felbomlása előtt az üzenetre reagálni.


AV
"A"
AV
"C"
SZGA
VEZÉR-
ÁLLOMÁS
AV
"B"
AV
"D"
Lekérdezési
sorrend pl.: A; B; A; C; A; D. . .

*8. ábra*
*Terminálok felfűzése névsor szerinti lekérdezéshez*


AV
AV
"B" VÁROS
TRANZIT
KÖZPONT
"D"
VÁROS
AV
AV
SZGA
HELYI
KAPCSOLÓ
KÖZPONT
TÖRZS-
VONA-
LAK
HELYI
KAPCSOLÓ
KÖZPONT
"A" VÁROS
TRANZIT
KÖZPONT
AV
AV
"C" VÁROS

*9. ábra*
*Többsíku vonalkapcsolt hálózat*

Legtöbbször azonban az *"adás vége"* jelzés (End of transmission: EOT) bontja az összekapcsolást, alaphelyzetbe hozza az összes adatállomást, és a vezérállomás csak akkor válaszolhat, ha sorrakerül (ld. 6. és 7. fejezet).

### *4.3. Kapcsolt összeköttetések*

Egy bonyolult távadatfeldolgozó rendszer sok adatállomásának változó hálózatigényéhez igen rugalmasan illeszkednek a jól szervezett kapcsolt hálózatok.

Az adatállomások között a kapcsolat valamilyen *"felhivási eljárás"* alkalmazásának eredményeként - például *"tárcsázással"*, illetve ennek elektronikus leutánzásával - épül fel.

A sokpontos kapcsolt adathálózat elsősorban kétoldalu összeköttetések létrehozására szolgál, de segitségével általában megoldhatók - további adatállomásoknak az adott összeköttetésre való felkapcsolásával - a konferencia-jellegü és körözvényadás jellegü többoldalu összeköttetések is. A kapcsolt hálózatok az összekötő áramkörök *jobb kihasználása érdekében* sok esetben *többsikuak*. Hálózati sikon a hálózat egyenrangu *csomópontjainak* halmazát értjük. A többsiku hálózatok igen kiterjedtek lehetnek, országos, nemzetközi, sőt világméretü adatátviteli hálózatok is léteznek.

A hálózat *legalsó sikját* a hálózat *előfizetői, felhasználói* végpontjai, az *előfizetői adatállomások* (EA) jelentik. Ezek lehetnek *számitógépes adatállomások is.*

A további sikokat a különböző hierarchiai magasságokban elhelyezkedő kapcsoló központok alkotják. A többsiku kapcsolt adathálózatok létrehozását számos tényező indokolja és felépitését sok szempont befolyásolja.

Több hálózati sik segitségével mód nyilhat a kihasználatlan vonalak rövidre választására, jól kihasznált *törzsvonalakba* történő összefogására, vonalhibák esetén az adatforgalom átterelésére, a hálózatágak és a különböző tipusu adatállomások sebességének illesztésére.

A kapcsolóközpontok elhelyezése, mérete, száma és a hálózat vonalai megválaszthatók ugy, hogy a teljes hálózat költségei, megbizhatósága a lehető legkedvezőbbek legyenek.

A kapcsolt hálózatok két adatállomása között az összeköttetés több szakaszból épül fel. A kapcsolás maga időigényes és a kapcsolások számának növelése csökkenti a rendszer megbizhatóságát. Nyilvánvaló követelmény, hogy a magasabb hálózati sikban elhelyezkedő kapcsoló központok és a hozzájuk tartozó hálózatágak megbizhatósága sokkal nagyobb legyen, mint az alacsonyabb sikok megbizhatósága.

A kapcsolt adathálózatok csak nagyon ritkán épülnek fel nemhierarchikus módon, mivel ebben az esetben a hálózat vezérlési, szervezési rendszere bonyolultabb és nehezebben tervezhető.

A hierarchikus felépitésü hálózatok vezérlési rendszere és igy optimalizálása is sokkal jobban kézben tartható.

Röviden foglaljuk össze a vonalkapcsolt és az üzenetkapcsolt hálózatok általános tulajdonságait.

### *4.3.1. Vonalkapcsolt hálózatok jellemzői*

A hagyományos postai hirközlő rendszerekben a két és többsiku *vonalkapcsolt hálózatok* széles körben elterjedtek, és ezeket az adatátvitelben egyre gyorsuló üzemben használják.

Egy többsiku vonalkapcsolt hálózat strukturáját a 9. ábra mutatja.

A vonalkapcsolt adathálózatoknak a vonalkapcsoló központokból eredő általános tulajdonságai a következők:

a) az összekapcsolt adatállomások között az összekapcsolás teljes idejében közvetlen a kapcsolat;

b) a kapcsolat felépitése - több részüzenetből álló információcsere esetén is - egyszeri, előzetes cimzéssel, "felhivással" történik, torlódásos alaprendszerben;

c) a teljes adatátviteli ut minden szakasza az összekapcsolás egész idejében foglalt;

d) a hálózat "átlátszó", körülményes a különböző sebességü, jelkészletü, üzenetformátumu adatállomások illesztése; az illesztés legtöbbször magukban az adatállomásokban történik;

e) a vonalszakaszok összekapcsolása *viszonylag hosszu időt* igényel, és az átvitel késleltetése ezenkivül még a vonalszakaszok futási idejétől is függ;

f) a forgalomátterelés - a még kevéssé elterjedt számitógépes vezérlés kivételével - nehézkes és csak előre rögzitett irányitási terv szerint történhet;

g) elsősorban hosszu üzenetek továbbitására gazdaságos;

h) a törzsvonalak átlagos kihasználtsága közepes, legfeljebb 60-70 %;

i) a vonalkapcsolás akár tér-, akár idő-, akár frekvenciaosztásban könnyen megoldható.

Különösen érdekes e tulajdonságok összevetése a később ismertetett üzenetkapcsoló hálózatok jellemzőivel.

A ma használatos vonalkapcsoló központok elsősorban mechanikus vagy elektromechnaikus elemekből épülnek fel. Csak az utóbbi időben kezdenek elterjedni a számitógéppel vezérelt, reed relékkel kapcsolt vagy tisztán elektronikus, félvezető áramkörös konstrukciók.

Ezekben a vonalkapcsoló központokban egy központi számitógép közvetlenül irányitja a vonalkapcsoló egységet. Minden befutó vonalat kiválasztó egység(selector) utján periódikusan, nagy sebességgel letapogat a központ. A processzor csak azokkal a vonalakkal foglalkozik, amelyek összeköttetést keresnek, majd elosztó egység (distributor) segitségével létrehozza az összeköttetést.

A megbizhatóság fokozása érdekében igen gyakran a teljes berendezés duplikált.

Bár a számitógéppel vezérelt vonalkapcsoló központok jelentős uj beruházást igényelnek, müködésük gazdaságos, és számos olyan előnyt nyujtanak, amelyeket az adatátviteli hálózatok jól hasznosithatnak. A programvezérlés rendkivül flexibilissé teszi a rendszert. Az alberendezések kitünően szabványosithatók, hiszen a változtatások egy részét programmal és nem a gépi berendezés módositásával végezhetjük el. A berendezés egyszerüen bővithető. Nagyon könnyen megvalósitható a hivószámok átszámitása, kerülőutak automatikus biztositása, az automatikus ujrahivás, az egymással nagy üzenetforgalmat lebonyolitó adatállomások közötti hivás-röviditès, több adatállomás egyidejü felkapcsolása, a "konferencia hivás" és a "körözvényadás". Az egyes speciális funkcióknak az adatvonalakhoz történő hozzárendelése időleges és könnyen változtatható.

E rengeteg kényelmes szolgáltatás mellett még a kapcsolásfelépitési idők is lerövidülnek és növekszik az adatvonalak kihasználtsága.

Ilyen rendszer például az NSZK-beli EDS szolgáltatás, amely alacsony-, közepes- és nagysebességü vonalakat bocsát az előfizetők rendelkezésére. /Irod. 2., 8./

A vonalkapcsolás nagy hátránya, hogy rövid üzenetek nagy sebességgel történő továbbitásakor a kapcsolásfelépitési idők hosszabbak lehetnek a tényleges üzenettovábbitási időknél.

### *4.3.2. Üzenetkapcsolt hálózat tulajdonságai*

Különösen a nyilvános postai hálózatoktól független, nagykiterjedésü adatátviteli hálózatok céljára használják előszeretettel az üzenetkapcsoló hálózatokat.

Mégegyszer emlékeztetünk arra, hogy a vonalkapcsolás és üzenetkapcsolás között az a leglényegesebb különbség, hogy az utóbbi esetben az adatcserét lebonyolitó adatállomások soha sem kerülnek közvetlen fizikai kapcsolatba egymással.

Az üzenetkapcsolás célja az adatvonalak jobb kihasználása, a vonalköltségek csökkentése. Az üzenetkapcsoló központ a vonalak maximális adatátviteli sebességével adhatja tovább a lassu adatállomásokról, az embertől származó összegyüjtött üzeneteket. Különösen a földrajzilag kiterjedt hálózatok törzsvonalait hasznosithatjuk igy előnyösen, ez az egyik oka annak, hogy a manuális, félautomatikus és automatikus üzenetkapcsolást az Amerikai Egyesült Államokban már a hagyományos távirótechnikában is elterjedten és gazdaságosan alkalmazták.

Ma is számos olyan számitógéppel vezérelt, real-time üzenetkapcsoló hálózat létezik, amelyet elsősorban a nem számitógépes feldolgozást, hanem csupán üzenettovábbitást folytató felhasználóknak épitettek ki.

Ilyen például a svájci automatikus távirószolgálat (ATECO).

Az üzenetkapcsolás azonban egyre szélesebb teret kér magának az adatállomások, mindenekelőtt az adatvégállomások közötti forgalom lebonyolitásában is.

Üzenetkapcsoló hálózatnál a "felhivási eljárás" az üzenetek *fejléccel* való ellátásából és a fejlécnek a kapcsoló központban történő *analiziséből* áll (boriték-elv).

Átmeneti üzenettárolásra a számitógépes üzenetkapcsoló központ részben operativ tárát, részben háttértárolóit használja. A beérkezett üzenetek az operativ tár *input puffer* tartományába, majd innen a gyors, közvetlen hozzáférésü mágneslemezes vagy mágnesdobos háttértárolóra kerülnek. Az üzenetfeldolgozó program a rendeltetési helyek és prioritások alapján *várakozási sorokba* rendezi az üzeneteket, amelyek azután az operativ tároló *output puffer* tartományába és az adatvonalakra jutnak (ld. 10. ábra). A legegyszerübb esetben az üzenetek továbbitása a beérkezési sorrend szerint történik (First-in-first-out üzenetkapcsolás).

Az üzenetekről lassubb elérésü háttértárolókon - leggyakrabban mágnesszalagon - naplót vezet, statisztikát készit a rendszer. Külön kérésre bizonyos üzeneteket a sikeres továbbitás után is megőrizhetünk.

A számitógépes üzenetkapcsoló központ csak az általa hibátlannak itélt üzeneteket fogadja el és továbbitja, átviteli hiba észlelésekor automatikus ismétléskérés, illetve automatikus ujratovábbitás történik. A rendszer a fentieken kivül sok járulékos feladatot is elláthat. /Irod. 4., 8., 13./

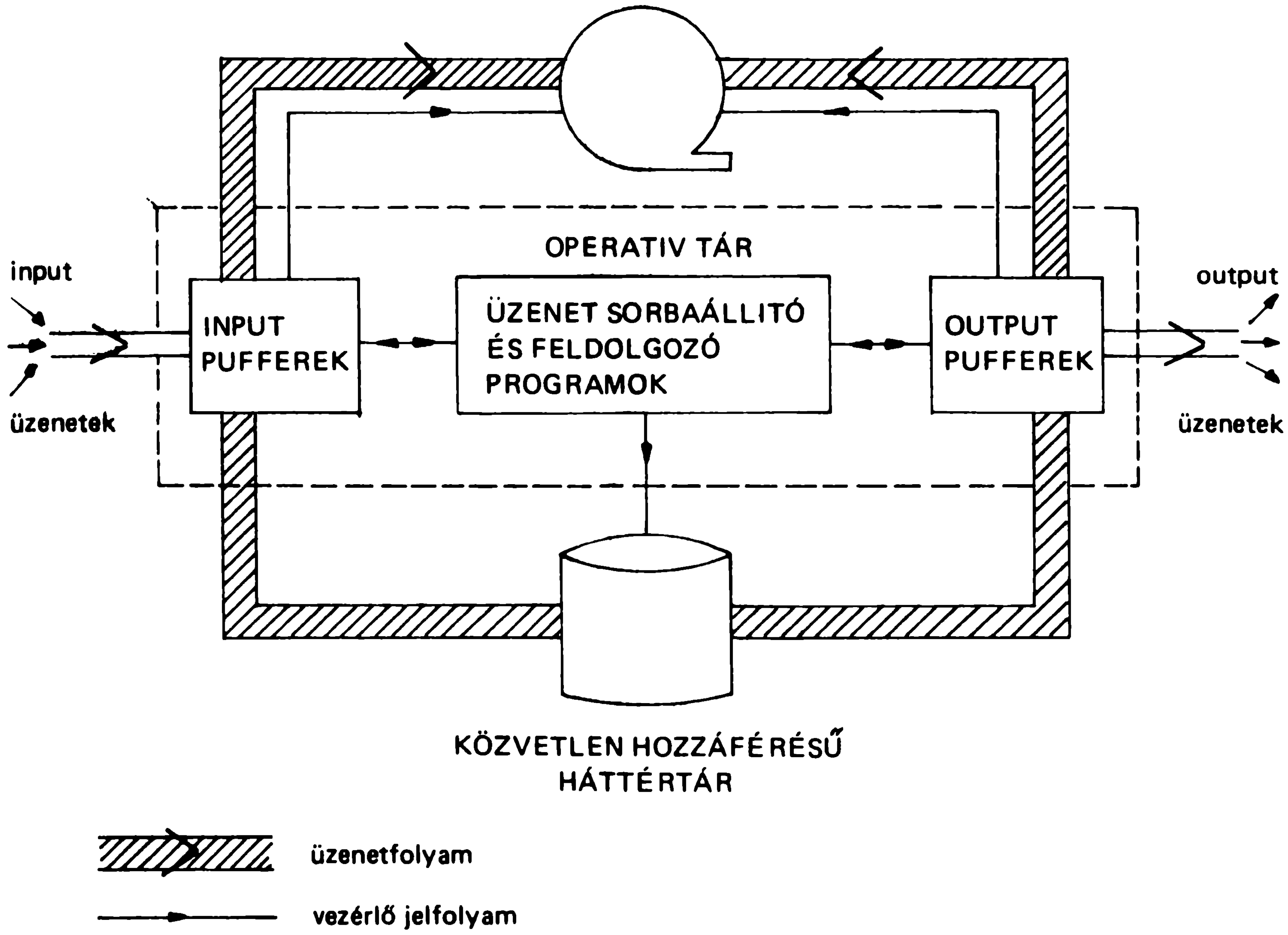
A számitógépes üzenetkapcsoló hálózatok általános tulajdonságai a következők:

a) Bár az összekapcsolt adatállomások között soha nincs közvetlen kapcsolat, a müködés lehet olyan gyors, hogy a közvetlenség érzete fennáll.

b) A kapcsolatok felépitése üzenetenként, az üzenet fejlécében elhelyezett cimek alapján történik, várakozásos alaprendszerben. Egy üzenet általában több cimzettnek is szólhat, a konferencia hivás és a körözvényadás is könnyen megoldható. Több üzenet közös cimzésére nincs lehetőség.

c) Az átviteli ut egyes *szakaszai időben eltolva foglaltak*. Nem szükséges, hogy a hivott adatállomás szabad legyen, az üzenet továbbitható, ha az adó adatállomástól az első puffertárolóig szabad az ut.

d) A különböző sebességü, jelkészletü és üzenetformátumu adatállomások illesztése az üzenetkapcsoló központban igen rugalmasan, programozás segitségével elvégezhető.


OPERATIV TÁR
input
üzenetek
INPUT PUFFEREK
ÜZENET SORBAÁLLITÓ ÉS FELDOLGOZÓ PROGRAMOK
OUTPUT PUFFEREK
output
üzenetek
KÖZVETLEN HOZZÁFÉRÉSŰ HÁTTÉRTÁR
üzenetfolyam
vezérlő jelfolyam

*10. ábra*
*Számitógépes üzenetkapcsolás egyszerüsitett sémája*

e) Az adatátviteli hibák kezelése egyszerű, leggyakoribb a hibás üzenetek automatikus megismételtetése.

f) A vonalszakaszok felkapcsolása általában igen gyorsan, elektronikus sebességgel megtörténik, az átvitel késleltetése azonban a vonalszakaszok futási idején tul az átmeneti tárolási időkkel és az ujrairányitási időkkel is megnövekszik. Ez azonban nem jelenti általában azt, hogy egy adathalmaz lassabban éri el a rendeltetési helyet, mint vonalkapcsolás esetén.

g) A forgalomátterelés könnyen automatizálható, üzenetkapcsoló központonként áttervezhető. A forgalomátterelés automatikus utvonalválasztás segitségével történhet (több mint két csomópont között).

h) Elsősorban rövid üzenetek továbbitása gazdaságos.

i) A törzsvonalak átlagos kihasználtsága magas, 85-95 %.

j) A számitógépes üzem elsősorban "időosztásos" adatátvitel megszervezésére előnyös.

k) Az üzeneteken általában nem történik módositás, de szükség esetén bizonyos üzenettipusokon "feldolgozás" jellegü müveletek is könnyen végezhetők (pl. kódkonverzió, átszerkesztés).

l) Egyes üzenetek - külön kérésre - hosszabb ideig is tárolhatók, és később ismét felidézhetők.

m) Időegységre eső bizonyos üzenetmennyiség felett feltétlenül gazdaságosabb, mint a vonalkapcsolás.

Sok üzenetkapcsoló rendszerben nem engedhető meg, hogy az üzenetek továbbitási lehetősége egyetlen központi üzenetkapcsoló számitógép megbizhatóságától függjön. Ilyen esetekben duplikált, sőt háromszoros rendszereket is alkalmaznak.

Az üzenetkapcsoló hálózatok a vonalkapcsoltakhoz hasonlóan lehetnek többsikuak is, a mai rendszerek azonban általában csak két szintet alkotnak.

Nagyon előnyös, hogy bár az üzenetkapcsoló számitógép tulajdonképpen az adatátviteli hálózat eleme, nincs akadálya annak, hogy e feladat ellátása mellett, kihasználatlan munkaidejében - amellyel főleg multiplikált rendszerek számitógépei rendelkeznek - a megfelelő operációs rendszerrel ellátott számitógép feldolgozó munkákat is végezzen.

Az üzenet-kapcsolás speciális esetére, az *üzenet-csomag-kapcsolásra* (packet switching) még visszatérünk (ld. 11. fejezet).

## 4.4. *Adathálózatok igénybevételi módozatai*

A távadatfeldolgozó hálózatok legnagyobb része ma még a hagyományos távközléstechnikában is alkalmazott vonalfajtákat használja adatátvitelre.

Az adathálózatok igénybevételük módja szerint két nagyobb csoportba, a *nyilvános adathálózatok* és a *független adathálózatok* csoportjába sorolhatók.

A nyilvános és az ugynevezett *bérelt* független hálózatokat adatátvitel céljára hazánkban a Magyar Posta a DATEL-szolgálat keretében bocsátja az előfizetők, illetve a bérlők rendelkezésére.

### *4.4.1. Nyilvános adathálózatok*

A meglévő nyilvános adathálózatok tulnyomó többsége az egyes országok postai szerveinek tulajdonát képező, hagyományos vonalkapcsolt távközlési hálózat.

E hálózatoknak kézenfekvő előnyei vannak:

1) Közvetlenül, vagy csekély járulékos beruházással alkalmasak adatátvitelre.

2) Nyilvánosak, s igy bármely felhasználó számára könnyen hozzáférhetők.

3) Széles körben elterjedtek.

4) Nagy távolságok áthidalására alkalmasak.

5) Szabványos eljárások, üzemmódok, berendezések, csatlakozások jellemzik.

6) Üzemeltetésük és karbantartásuk jól szervezett.

7) Hálózatágak meghibásodása vagy tulterheltsége esetén kerülőutak viszonylag könnyen biztosithatók.

8) Kisforgalmu adatállomások esetében gazdaságosak; mivel felépitésük hierarchikus, általában csak az előfizetői sik lesz kihasználatlan.

E nyilvánvaló - és a hátrányokat jelenleg még kompenzáló - előnyök mellett azonban kedvezőtlen tényezőkkel is számolnunk kell, amelyek abból fakadnak, hogy az ilyen hálózatokat eredetileg nem számitógépes adatátvitel céljára tervezték.

Világszerte óriási erőfeszitések történnek a hagyományos távközlési hálózatok adatátviteli tulajdonságainak javitására, de - bár ezek általában sikerrel járnak - egyre több, kimondottan adatátvitelre tervezett nyilvános adathálózatot épitenek, hogy a növekvő igényeket magasabb szinten elégithessék ki.

A hagyományos távíró, telex és távbeszélő hálózat használatának közös korlátozó tényezője, hogy:

1) A kapcsolásfelépitési idők hosszuak.

2) A hálózatok adatátviteli hibaaránya sok esetben nem kielégitő, megbizhatóságuk csak igen nagy ráforditással lenne megfelelő mértékben fokozható.

3) Adatátviteli sebességük sokszor nem éri el a távadatfeldolgozó rendszer követelményeit, a nyilvános hálózatoknak még a bérelt független hálózatokkal azonos tipusu vonalai is általában ez utóbbiaknál csak kisebb sebességgel vehetők igénybe.

4) A hálózat külső és belső zavarokra is meglehetősen érzékeny.

5) A hálózat általában nem képes a hivás jogosságának vizsgálatára, a titkosság követelménye nehezen kielégithető.

6) A hálózat teljesen "átlátszó", és igy nem képes az igényekhez rugalmasan illeszkedő sebesség-, kód-, illetve formátum-átalakitásra.

7) A hagyományos hálózat rendszerint alkalmatlan egyszerre több adatállomás felkapcsolására, a körözvényadásra, pedig ez sok távadatfeldolgozó rendszerben követelmény lenne.

8) Az üzemmódok, adatátviteli berendezések, jelspecifikáció stb. nemzetközileg nem kielégitő mértékben szabványositottak, az egyes országok szabványai eltérőek, igy a földrajzi kiterjesztés jelentős járulékos beruházást igényelhet.

a) A *nyilvános táviróhálózat* távadatfeldolgozási alkalmazása jelentéktelen mértékü. Ennek oka elsősorban az, hogy a postai táviróvonalak - bár kissebességü, tipikusan 50 baud-os START-STOP átvitelre alkalmasak - a postahivatalokban végződnek, ami adásnál (feladás) és vételnél (kézbesités) egyaránt nagy késleltetést jelent. Ennek ellenére off-line kapcsolatra kevéssé igényes esetekben a nyilvános táviróhálózat elvileg megfelel. A postahivatalok általában csak szövegesen megadott információt fogadnak el továbbitásra, de néhány országban lyukszalagról-lyukszalagra történő továbbitást is vállalnak.

b) A *nyilvános telexhálózat* - amely tulajdonképpen előfizetőkhöz vezetett táviróhálózat - már sokkal nagyobb jelentőségü a távadatfeldolgozásban.

A nyilvános telexhálózat nemzetközi méretü, legalább országos szinten teljesen automatizált, vonalkapcsolt digitális távgépiró hálózat. A telex-irógépek közvetlenül alkalmasak adatbevitelre és adatmegjelenitésre, sőt legtöbbször lyukszalag-olvasó és lyukszalaglyukasztó egységekkel is fel vannak szerelve.

A telex-hálózat használatát korlátozza, hogy kötött kódban (CCITT. No. 2. ABC, vagyis Nemzetközi 2-es számu Távirό ABC) dolgozik, adatátviteli sebessége alacsony, rendszerint nem több, mint 50 baud, és megbizhatósága sem kielégitő, bit hibaaránya -mely erősen függ a vonalszakaszok számától - $10^{-4}$ és $10^{-6}$ közé esik.

A telex-állomás rendszerint off-line kapcsolódik az adatfeldolgozó központhoz, de a telex-hálózat, megfelelő kiegészitő berendezések segitségével közvetlenül számitógéphez is csatlakoztatható. Ebben az esetben programmal vagy hardware-rel le kell utánozni a telexállomás müködését.

Bár a telexhálózatok egy része duplex összeköttetésre képes, rendszerint félduplex összeköttetésre használják.

c) A *nyilvános távbeszélő hálózatok* a telex hálózatoknál nagyobb kiépitettségüek és jelentőségük is sokkal nagyobb a távadatfeldolgozásban.

Automatizáltságuk foka azonban a legtöbb országban még alacsonyabb, általában csak a helyi hálózatrészek szintjén automatizáltak, országos és nemzetközi automatikus távkapcsolás csak a legfejlettebb ipari országokban van.

Igy mindenekelőtt a távválasztás automatizálását kell megoldani, mivel a manuális kapcsolás nagyon lassu, és megbizhatatlanná teszi az adatátvitelt.

A távbeszélő hálózat hangfrekvenciás jelek analóg átvitelére készült. Az adatállomásokhoz történő csatlakoztatása nemzetközi távközlési szervek által szabványositott és az országos postai szervek által engedélyezett, esetleg a postai szolgáltatás részét képező modemekkel történik. A szabványokban engedélyezett modemek választéka sok esetben nem kielégitő, ezért nyilvános telefonhálózaton néha nehézségbe ütközik az adatállomások megfelelő hatásfoku illesztése.

Az adatállomások és az ezek részét képező modemek kialakitása szerint a nyilvános távbeszélő hálózat szimplex, félduplex és duplex összeköttetés létrehozására egyaránt használható.

Egyrészt az adatátvitel igen "intenziv igénybevétel" a távbeszélő áramkörök számára, másrészt a régi épitésü távbeszélő áramkörök müszaki paraméterei is leromlottak, ezért a nyilvános telefonhálózatokon általában, - igy hazánkban is - 1200 bit/sec a megengedett maximális adatátviteli sebesség. Sok vonalszakaszon ennél nagyobb adatátviteli sebesség is lehetséges volna, más szakaszokon viszont nagyobb sebesség esetén erősen leromlik az átvitel megbizhatósága.

Sajnos, a nyilvános távbeszélő hálózatok megbizhatósága még a telex-hálózatokénál is alacsonyabb, bit-hibaarányuk $10^{-3}$-$10^{-5}$, igy a legtöbb esetben bonyolultabb hibavédelem alkalmazása szükséges.

d) *Uj nyilvános adatátviteli hálózatok*

Világszerte egyre több nyilvános távközlési hálózatot létesitenek, kimondottan adatátvitel céljára.

Az adatátvitelre tervezett uj hálózatok egy része távbeszélő tipusu analóg hálózatrendszer. Ezek a hagyományos rendszereknél nagyobb megbizhatóságuak, nagyobb sebességüek, kapcsolási idejük kisebb, távválasztásuk eleve automatikus, de nem illeszkednek az Egységes Digitális Távközlési Hálózat elképzelésébe.

Nagyobb jelentősége van a nyilvános digitális hálózatoknak. Ilyen nyilvános hálózat a már emlitett DATEX-hálózat is, melyet az előfizetők a DATEL-szolgálat keretében használhatnak. A DATEX-hálózat táviró-tipusu első szintje gyors kapcsolási idejü, egyszerü vonalcsatlakozást és duplex összeköttetést lehetővé tevő, 200 baud maximális sebességü, transzparens hálózat. Sajnos, hazánkban még csak szük körben terjedt el és a hozzátartozó adatállomási berendezések még nehezen beszerezhetők. A DATEX-szolgálat nagyobb sebességü változatai (4800 bit/sec, 9600 bit/sec, 48 Kbit/sec) néhány országban már müködnek, hazánkban is történtek lépések a közép-sebességü Datex-szolgáltatás helyi viszonylatban történő beinditására. /Irod. 7./

### 4.4.2. *Független adathálózatok*

Függetlennek nevezünk minden olyan adathálózatot, amely a nyilvános adathálózatokkal kölcsönös adatforgalmat lebonyolitani nem tud, a bárki számára elérhető, nyilvános adathálózatoktól függetlenül üzemel. A független adathálózat lehet a postától *bérelt* vagy az adott távadatfeldolgozó rendszert üzemeltető vállalat, intézmény, szervezet *saját tulajdonát képező* elkülönült hálózat.

Már a távadatfeldolgozó rendszerek kiépitésének kezdeti időszakában is jelentkeztek olyan követelmények, amelyek csak független adathálózattal voltak kielégitőek.

A független adathálózatok előnyei, a nyilvános hálózatokhoz képest, abból következnek, hogy a független adathálózatokat mindig kimondottan adatátviteli célra, legtöbbször konkrét adatátviteli feladattipusokra vagy feladatokra tervezik.

A független adathálózatok előnyei a következők:

1) Állandóan a felhasználó rendelkezésére állnak.

2) Nincs kitéve az adott távadatfeldolgozó rendszerbe nem tartozó felhasználók zavaró hatásának, a titkositás könnyen biztositható.

3) A hálózat hibaaránya alacsonyra tervezhető, megnő az adatátvitel biztonsága.

4) A nyilvános adathálózatokéval azonos vonaltipusok általában nagyobb sebességüek, például bérelt telefonvonalakon elérhető a 4800 bit/sec, sőt a 9600 bit/sec adatátviteli sebesség is.

5) Az egyes vonalszakaszok és a hozzájuk tartozó adatállomási berendezések minden szempontból jól illeszthetők egymáshoz.

6) Az összeköttetések létrehozásának ideje vonalszakaszonként az igényeknek megfelelően, szükség szerint alacsonyra tervezhető.

7) Adott nagyságu adatforgalom felett - hálózattipustól függően - gazdaságosabb, mint a nyilvános adathálózat.

8) A hálózatba betervezhető az igényekhez illeszkedő sebesség, kód- és formátum-átalakitás, ugyanakkor a hálózat "átlátszó"-ként is üzemelhet.

9) Szabványok kevésbé korlátozzák a választott megoldást.

Ezek mellett az előnyök mellett figyelembe kell vennünk a következő hátrányokat is:

1) A független adathálózat létrehozása nagy volumenü uj beruházást jelent, a bérleti vonalköltségek is magasak.

2) A hálózat fenntartása általában nem oldható meg olyan hatékonyan, mint a nagy postai szervezetekben. Ha a bérelt hálózat karbantartását a bérbeadó végzi, ez lényeges többletkiadást jelent.

3) A hálózat gazdaságos üzemeltetését biztositó mértékü kihasználtság csak igen körültekintő szervezéssel érhető el.

4) Alacsony adatforgalmi szint esetén általában kevésbé gazdaságos, mint a nyilvános hálózatok.

5) Kerülőutak biztositása rendszerint gazdaságtalan, a kerülőutak nélküli hálózat viszont nagyon letörés-érzékeny a hálózatelemek meghibásodására.

A független adathálózatokban az eddig megismert összes hálózatstrukturák és vonaltipusok megtalálhatók, bár arányuk más, mint a nyilvános adathálózatokban.

Például a vonalkapcsolt független hálózatok igen ritkák a vonalkapcsolt nyilvános hálózatokhoz képest, ugyanakkor nagyon gyakran találkozhatunk a felfüzéses független adathálózatokkal.

Szélessávu vonalak nyilvános szolgáltatás formájában csak néhány országban elérhetők, annál gyakoribbak a szélessávu, bérelt vagy vállalati tulajdonban lévő független hálózatágak. In-plant vállalati belső távadatfeldolgozó rendszerekben leggyakrabban a vállalat saját tulajdonát képező független hálózatokat alkalmazzák.

# 5.

# Kódolás és hibakorlátozás

Az adatátvitel során két kiválasztott, egymással kommunikáló adatállomás egy adott pillanatban adó (forrás) és vevő (nyelő) relációban jelenik meg az átviteli rendszer két végpontján. Az adó (vevő) véges számu egymástól különböző jel, karakter kibocsátására (fogadására) képest, ezért információelméleti szempontból megfelel a *diszkrét forrásnak* (nyelőnek).

Az adatok továbbitására analóg és digitális vonalakat egyaránt felhasználhatunk. Az analóg jeleket is digitális jelekből nyerjük és ilyenekké alakitjuk vissza, ha tehát az átalakitókat (modemeket) is beleértjük a csatornába, akkor végül is digitális csatornához jutunk. Ez véges számu digitális elemet, csatornaszimbólumot (bit, dibit stb.) képes megkülönböztetni és továbbitani, tehát *diszkrét csatorna*.

Az adatátvitelhez használt diszkrét források, nyelők és csatornák információelméleti leirásánál egyszerüsitő feltételek bevezetésével a valóságot jól megközelitő *modelleket* alkotunk, és a gyakorlatban a modellekben matematikailag megfogalmazott tulajdonságu *rendszerelemek* optimális együttmüködtetésére, *illesztésére* törekszünk. Itt csak arra vállalkozhatunk, hogy röviden (és heurisztikusan) megkiséreljük összefoglalni a modellalkotás motivumait, majd valamivel bővebben megvizsgáljuk a - csatornamodell által igényelt, és a számitógépes szakember számára szokatlan mértékü - hibakorlátozás kérdéseit.

## 5.1. *A modellalkotás néhány kérdése, kódolás*

Az adó (a továbbiakban csak a forrás-oldalt vizsgáljuk, a vevőre értelemszerüen igazak az adó tulajdonságai) és az adatcsatorna diszkrét jellege minden megszoritás nélkül igaz, a modellalkotáshoz szükséges további megállapitások azonban csupán (jól) közelitő feltételezések. /Irod. 11., 12./

A források jelkészlete és a csatornaszimbólumok eltérnek egymástól, ezért szükség van ezek egyértelmü összerendelésére. A gyakorlatban a csatornaszimbólumok száma kisebb, mint a forráselemek száma, ezért az egyes forráselemeknek csatornaszimbólum-sorozatokat feleltetünk meg. Ez az eljárás a *forráskódolás*. Az egyes forráselemekhez rendelt csatornaszimbólum sorozatok a *kódszavak*.

A gyakorlatban a forrásról feltételezzük, hogy "memóriátlan". Ez azt jelenti, hogy a forrás nem emlékszik a saját előéletére, tehát kibocsátott jelsorozatának n-edik eleme független az őt megelőző (n-1) elem mindegyikétől. Ez a feltételezés numerikus információ esetében igen jó közelités. Alfabetikus információ, értelmes szöveg adásakor azonban a betük több-kevesebb korrelációban vannak egymással, tehát bizonyos jelek egyáltalán nem, vagy alig hordoznak információt. A forrás emlékező tulajdonsága elvileg kihasználható lenne információ-tömöritésre, s igy az adatátviteli sebesség növelésére, ezt a lehetőséget azonban bonyolult kiaknázhatósága miatt nem hasznositjuk.

Ugyancsak feltételezzük, hogy a forráselemek azonos valószinüséggel jelennek meg a forrás kimenetén. Szöveges információnál ezzel szemben bizonyos betük és betücsoportok igen gyakoriak. Kézenfekvőnek tünne a gyakori szimbólumokat rövidebb, a ritkán előfordulókat hosszabb idő alatt továbbitani. Igy az átlagos adatátviteli sebesség, az időarányok megfelelő választása esetén, jelentősen megnőhetne. Ezzel az elvi lehetőséggel a gyakorlatban nem élünk, hanem *feltételezzük*, hogy minden forráselem azonos valószinüséggel fordul elő, vagyis, hogy *a forráselemek által átlagosan hordozott információ*, az *entrópia maximális*. Ennek a feltételezésnek praktikus okai vannak. Ha ugyanis a nagy valószinüségü forráselemekhez gyorsan továbbitható rövid kódszavakat választanánk, mig a kisebb valószinüségü elemekhez arányosan hosszabbakat, akkor ügyelni kellene a kódszavak irreducibilitására, vagyis arra, hogy az egyes kódszavak elválasztó szimbólumok nélkül is egyértelmüen megfejthetők legyenek. A változó hosszuságu kódszavak bonyolultabb és flexibilisebb fogadó, dekódoló és tároló rendszert igényelnének.

A (forrás)kódolásnál tehát állandó hosszuságu kódszavakat alkalmazunk.

A digitális csatorna igen lényeges tulajdonsága, hogy átvitele a korábban átvitt jelektől függetlennek tekinthető, a *csatorna* is *memóriátlan*. A valóságos csatornák azonban a bemenetükre adott jelet csak 1-nél kisebb valószinüséggel jelenitik meg a kimenetükön, vagyis a *csatorna zajos*, az *adatátvitel nem feltétlenül sikerül hibátlanul*. Az ilyen zajos, diszkrét, memóriátlan csatornát információelméleti szempontból akkor ismerjük, ha pontosan tudjuk, hogy melyik bemenő csatornaszimbólumból milyen valószinüséggel melyik kimenő csatornaszimbólum jön létre az adatátvitel során. Ugy tekintjük, hogy ezek az átmeneti valószinüségek is függetlenek a csatorna előéletétől.

Az adatvonalakkal realizált valódi adatcsatornák szimbólumtévesztési valószinüsége tulságosan nagy és technológiai megoldásokkal csak igen jelentős költségráforditások árán, de nem tetszőleges mértékben csökkenthető. *Az adatátvitel megbizhatósága általában 2-4 nagyságrenddel kisebb, mint a számitástechnikai elemek meg-*

*bizhatósága*, ezért az *adatátvitelnél általában szükség van a nagyfoku hibakorlátozásra*. A hibakorlátozás módszereit az információelmélet és a hibakorlátozó kódok elmélete mutatja meg.

A hibakorlátozás alapelve: az ideális esetben szükségeshez képest felesleges tényezőket, *redundanciát* viszünk a rendszerbe.

A redundancia bevitelének közvetlen és kézenfekvő módszere az, hogy a forráselemek megjelenitéséhez feltétlenül szükségesnél hosszabb kódszavakat választunk. Ezzel tulajdonképpen a kódolt forráselemeket ujra kódoljuk, *hibakorlátozó kódolást* végzünk. Az uj, hosszu kódszavaknak a megjelenitésen tulmenő feladata az, hogy bizonyos előre meghatározott szabályoknak is eleget tegyenek. A megjelenités eltévesztése egyuttal a szabály áthágását is jelenti, igy a hiba felismerhető. A módszer kiterjeszthető ugy, hogy a többletinformációt, a redundanciát nem az egyes kódszavakhoz, hanem ezek sorozataihoz, tehát forráselem-blokkokhoz rendeljük.

Az egyes forráselemekhez és ezek blokkjaihoz egyszerre is tartozhatnak redundáns megjelenitő elemek.

A redundancia más uton is bejuthat a rendszerbe. Tulajdonképpen redundanciát jelent az is, ha az ideális átvitel szempontjából felesleges adatcsatorná(ka)t használunk. A "felesleges" csatorna létrejöhet az előzőleg használt csatorna ismételt felhasználásával, vagy egy fizikailag másik csatorna használatával.

A gyakorlatban hibakorlátozásra általában a vázolt redundancia beviteli módszerek kombinációját alkalmazzák.

A fentiekből látszik, hogy egyetlen forrás és nyelő közötti adatátvitelhez az *adóoldalon* általában *forráskódolásra* és *hibakorlátozó kódolásra*, a vevőoldalon *hibafelismerésre* és *dekódolásra*, az eredeti forrásszimbólumok ujbóli megjelenitésére van szükség. Általában az adó visszajelzést is igényel, amelyből megtudja, hogy a vétel hibás vagy hibátlan volt-e. Csak ez alapján tud dönteni arról, hogy megismételje-e az adást.

Az adatállomásokon a forráselemek csaknem mindig eleve kódoltan, esetleg redundáns kód formájában jelennek meg, hiszen többnyire digitális készülékektől származnak. Sok esetben a forráskódolás és a hibakorlátozó kódolás egyetlen lépésben történik. Sokszor a vevőoldalon csak hibakorlátozó dekódolást kell végezni, mivel a vevő vagy kódolt formában továbbadja, vagy feldolgozza az információt.

A távadatfeldolgozó rendszerek adatátviteli hálózatában egyaránt elterjedtek a hagyományos adatátvitelben és adatfeldolgozásban használt szabványos kódok és az ujabb, távadatfeldolgozáshoz kidolgozott kódok is. Leggyakoribbak az ötbites nemzetközi távirókód (CCITT-code Nr. 2.) és ennek változatai (pl. a Baudot-kód), a karakterstrukturára épült 6-bites kódok (BCD, TTS, SBT), a hétbites ASCII kód és az ennek alapján elfogadott ISO kód (CCITT-code Nr. 5.), a byte strukturának megfelelő nyolcbites EBCDIC és a 8 bitre bővitett ASCII (ASCII-8), valamint ISO kódok.

A kódok többségében a szokásos alfanumerikus és speciális jeleken tul további *vezérlő karaktereket* is találunk. Ezek egy része *adatátviteli vezérlő karakter*, más része *funkcionális vezérlő karakter*. Segitségükkel az üzenetek továbbitásának vezérlését, az adatállomások megfelelő müködtetését és az egyéb járulékos feladatokat láthatjuk el (ld. 6. fejezet).

Nézzük meg a legfontosabb kódok táblázatait, és az alapvető vezérlő karakterek jelentését (ld. 4., 5., 6., 7., 8. táblázatok).

## *5.2. Hibakorlátozás*

Láttuk, hogy a forráskódok általában nem adhatók közvetlenül a zajos csatornára, mert ebben az esetben az igényeinknek megfelelő minőségü adatátvitel nem lenne biztositható.

A gyakorlati adatátviteli rendszerek megbizhatóságát valamilyen *hibakorlátozó kódolás* és *hibakorlátozási stratégia* megfelelő összehangolására törekedő alkalmazásával fokozzák.

Az, hogy egy tervezés alatt álló távadatfeldolgozó rendszer egyes adatátviteli szakaszaira milyen hibakorlátozási stratégiát és ehhez milyen hibakorlátozó kódolást válasszunk, a rendszer elemzője szempontjából egyik legbonyolultabb és legnehezebben megválaszolható kérdés.

Az egész integrált rendszer biztonsága szempontjából általában éppen az adatátviteli csatornák megbizhatósága a szük keresztmetszet, ezért hibakorlátozásuk különös gondot igényel.

Az adatállomások összekapcsolásánál az adatátvitel üzenetekkel, blokkjellegü egységekben történik. Az üzenetek lehetnek fix, vagy változó hosszuságuak, méretüket a feldolgozó rendszer jellemzőihez és az adatátviteli rendszer paramétereihez is illeszteni kell.

Az *üzenetek mérete*, a *csatornahibák fajtái és gyakorisága*, az *alkalmazott hibakorlátozó kód és hibakorlátozási stratégia* kölcsönösen összefüggnek egymással.

Ráadásul a hibakorlátozásnak jelentős költségkihatásai is vannak.

Az üzenetek hossza elsősorban a feldolgozás jellegétől függ. Ha a távadatfeldolgozást ember-gép dialógus jellemzi, akkor általában alacsony sebességü átviteli rendszer is megfelel és célszerü rövid - egy vagy néhány karakterből álló - blokkokat választani.

Nagy mennyiségü adatnak nagy sebességgel történő továbbitásakor hosszu üzeneteket használunk.

Ez történik például távolsági kötegelt adatfeldolgozás (remote batch processing) esetében. Az üzenetek szövegrésze itt rendszerint rekord méretü, ha például az adatforrás lyukkártya, akkor 80, illetve 160 karakterből állhat. Előfordul az is, hogy a továbbitandó adathalmazt - a blokkméret optimalizálása céljából - eredeti szerkezetétől teljesen független méretü üzenetekre bontják.

| CCITT SORSZÁM | | 32 | 5 | 28 | 31 | 27 | 20 | 1 | 9 | 14 | 15 | 19 | 18 | 8 | 4 | 12 | 26 | 21 | 3 | 13 | 6 | 7 | 10 | 16 | 23 | 2 | 25 | 11 | 22 | 24 | 30 | 17 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BITPOZÍCIÓK | 1 | | ● | | | | | ● | | | | ● | | | ● | | ● | ● | | | ● | | ● | | ● | ● | ● | ● | | ● | ● | ● | ● |
| | 2 | | | ● | | | | ● | ● | | | | ● | | | ● | | ● | ● | | | ● | ● | ● | ● | | | ● | ● | | ● | ● | ● |
| | | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| | 3 | | | | ● | | | | ● | ● | | ● | | ● | | | | ● | ● | ● | ● | | | ● | | | ● | ● | ● | ● | | ● | ● |
| | 4 | | | | | ● | | | | ● | ● | | ● | | ● | | | | ● | ● | ● | ● | ● | | | ● | | ● | ● | ● | ● | | ● |
| | 5 | | | | | | ● | | | | ● | | | ● | | ● | ● | | | ● | | ● | | ● | ● | ● | ● | | ● | ● | ● | ● | ● |
| CCITT Nr. 2. | | ⊖ | E | ≡ | BK | < | T | A | I | N | O | S | R | H | D | L | Z | U | C | M | F | G | J | P | W | B | Y | K | V | X | 1… | O | A… |
| | | ⊖ | 3 | | | | 5 | – | 8 | . | 9 | ' | 4 | -¦- ◥ | ✠ | ) | + | 7 | : | . | ◥ | ◥ | 🔔 | 0 | 2 | ? | 6 | ( | = | / | | 1 | |

☐ NINCS LYUKASZTÁS

⊡ VAN LYUKASZTÁS

< KOCSI VISSZA

≡ SOREMELÉS

🔔 CSENGETÉS

1… SZÁMVÁLTÓ

A…BETŰVÁLTÓ

BK…BETŰKÖZ

⊖ 32. KOMBINÁCIÓ

◥ KÜLÖNLEGES JELEKNEK FENNTARTVA

✠ KI VAN OTT?

*4. táblázat*
*CCITT Nr. 2. ABC*

| 7 | | | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | | | | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 5 | | | | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 4 | 3 | 2 | 1 | | | | | | | | |
| 0 | 0 | 0 | 0 | NUL | DLE | SP | 0 | @ * | P | * | p |
| 0 | 0 | 0 | 1 | SOH | DC$_1$ | ! | 1 | A | Q | a | q |
| 0 | 0 | 1 | 0 | STX | DC$_2$ | " * | 2 | B | R | b | r |
| 0 | 0 | 1 | 1 | ETX | DC$_3$ | £ * | 3 | C | S | c | s |
| 0 | 1 | 0 | 0 | EOT | DC$_4$ | $ * | 4 | D | T | d | t |
| 0 | 1 | 0 | 1 | ENQ | NAK | % | 5 | E | U | e | u |
| 0 | 1 | 1 | 0 | ACK | SYN | & | 6 | F | V | f | v |
| 0 | 1 | 1 | 1 | BEL | ETB | " * | 7 | G | W | g | w |
| 1 | 0 | 0 | 0 | FE$_0$ | CAN | ( | 8 | H | X | h | x |
| 1 | 0 | 0 | 1 | FE$_1$ | EM | ) | 9 | I | Y | i | y |
| 1 | 0 | 1 | 0 | FE$_2$ | SUB | * | : * | J | Z | j | z |
| 1 | 0 | 1 | 1 | FE$_3$ | ESC | + | ; * | K | [ * | k | * |
| 1 | 1 | 0 | 0 | FE$_4$ | IS$_4$ | , | < | L | * | l | * |
| 1 | 1 | 0 | 1 | FE$_5$ | IS$_3$ | – | = | M | ] * | m | * |
| 1 | 1 | 1 | 0 | SO | IS$_2$ | . | > | N | * | n | _ * |
| 1 | 1 | 1 | 1 | SI | IS$_1$ | / | ? | O | – | o | DEL |

Megj.: a *-al jelölt kódkombinációk jelentése változhat.

*5. táblázat*
*ISO 7-bites kód (CCITT. Nr. 5. ABC)*
*(Csaknem azonos az ASCII-7 kóddal)*

| | | | 8 | 0 | | | | 0 | | | | 1 | | | | 1 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 7 | 0 | | | | 1 | | | | 0 | | | | 1 | | | |
| | | | 6 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| | | | 5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 4 | 3 | 2 | 1 | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | NUL | DLE | DS | | SP | | – | | | | | | | | | 0 |
| 0 | 0 | 0 | 1 | SOH | DC1 | SOS | | | | / | | a | j | | | A | J | | 1 |
| 0 | 0 | 1 | 0 | STX | DC2 | FS | SYN | | | | | b | k | s | | B | K | S | 2 |
| 0 | 0 | 1 | 1 | ETX | TM | | | | | | | c | l | t | | C | L | T | 3 |
| 0 | 1 | 0 | 0 | PF | RES | BYP | PN | | | | | d | m | u | | D | M | U | 4 |
| 0 | 1 | 0 | 1 | HT | NL | LF | RS | | | | | e | n | v | | E | N | V | 5 |
| 0 | 1 | 1 | 0 | LC | BS | ETB | UC | | | | | f | o | w | | F | O | W | 6 |
| 0 | 1 | 1 | 1 | DEL | IL | ESC | EOT | | | | | g | p | x | | G | P | X | 7 |
| 1 | 0 | 0 | 0 | | CAN | | | | | | | h | q | y | | H | Q | Y | 8 |
| 1 | 0 | 0 | 1 | RLF | EM | | | | | | | i | r | z | | I | R | Z | 9 |
| 1 | 0 | 1 | 0 | SMM | CC | SM | | ¢ | ! | | : | | | | | | | | ≉ |
| 1 | 0 | 1 | 1 | VT | CU1 | CU2 | CU3 | · | $ | , | # | | | | | | | | |
| 1 | 1 | 0 | 0 | FF | IFS | | DC4 | < | * | % | a | | | | | ⑁ | | ⑀ | |
| 1 | 1 | 0 | 1 | CR | IGS | ENO | NAK | ( | ) | – | ' | | | | | | | | |
| 1 | 1 | 1 | 0 | SO | IRS | ACK | | + | ; | > | = | | | | | ⑂ | | | |
| 1 | 1 | 1 | 1 | SI | IUS | BEL | SUB | / | ¬ | ? | " | | | | | | | | EO |

*6. táblázat*
*EBCDIC jelkészlete*

| 8 | | | | 0 | | | | 0 | | | | 1 | | | | 1 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | | | | 0 | | | | 1 | | | | 0 | | | | 1 | | | |
| 6 | | | | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 5 | | | | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 4 | 3 | 2 | 1 | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | NUL | DLE | | | SPA | 0 | | | | | | P | | | | p |
| 0 | 0 | 0 | 1 | SOH | DC1 | | | ! | 1 | | | | | A | Q | | | a | q |
| 0 | 0 | 1 | 0 | STX | DC2 | | | " | 2 | | | | | B | R | | | b | r |
| 0 | 0 | 1 | 1 | ETX | DC3 | | | # | 3 | | | | | C | S | | | c | s |
| 0 | 1 | 0 | 0 | EOT | DC4 | | | $ | 4 | | | | | D | T | | | d | t |
| 0 | 1 | 0 | 1 | ENQ | NAK | | | % | 5 | | | | | E | U | | | e | u |
| 0 | 1 | 1 | 0 | ACK | SYN | | | & | 6 | | | | | F | V | | | f | v |
| 0 | 1 | 1 | 1 | BEL | ETB | | | ' | 7 | | | | | G | W | | | g | w |
| 1 | 0 | 0 | 0 | BS | CAN | | | ( | 8 | | | | | H | X | | | h | x |
| 1 | 0 | 0 | 1 | HT | EM | | | ) | 9 | | | | | I | Y | | | i | y |
| 1 | 0 | 1 | 0 | LF | SS | | | • | : | | | | | J | Z | | | j | z |
| 1 | 0 | 1 | 1 | VT | ESC | | | + | ; | | | | | K | [ | | | k | { |
| 1 | 1 | 0 | 0 | FF | FS | | | , | < | | | | | L | \ | | | l | / |
| 1 | 1 | 0 | 1 | CR | GS | | | − | = | | | | | M | ] | | | m | } |
| 1 | 1 | 1 | 0 | SO | RS | | | | > | | | | | N | ^ | | | n | ¬ |
| 1 | 1 | 1 | 1 | SI | US | | | / | ? | | | | | O | — | | | o | DEL |

*7. táblázat*
*ASCII-8 jelkészlete*

A *csatornahibák fajtái* nagyon jól leirhatók a *véletlen hiba* és a *hibacsomó* fogalmával. Véletlen hibáról akkor beszélünk, ha a csatornaszimbólumok a szimbólumsorozatban elfoglalt helyüktől függetlenül azonos valószinüséggel romlanak el. A hibacsomót ezzel szemben az jellemzi, hogy a csomósodás határai - az első és utolsó meghibásodott csatornaszimbólum - között elhelyezkedő szimbólumok meghibásodási valószinüsége (sokkal) nagyobb, mint az ezeken kivül eső szimbólumoké.

A gyakorlatban a hibacsomók főleg 2-3 stb. egymás melletti bit "azonosra változását" jelentik.

Az atmoszférikus zavarok elsősorban véletlen hibákat, a távbeszélő és táviró hálózat zavarai főleg hibacsomókat okoznak.

Általános elv, hogy a nagy hibaszázaléku csatornákon - a hibák fajtájától függetlenül - kerüljük a hosszu blokkok átvitelét.

A *hibakorlátozó kódoknak* még a felsorolására sem vállalkozhatunk e szükre szabott alfejezetben, csupán néhány gyakran alkalmazott bináris kódolási eljárás legfontosabb tulajdonságait foglaljuk össze.

A hibakorlátozó kódolás olyan müvelet, amely a kódolt adatelemek célszerüen választott sorozatát tudatosan és meghatározott törvényszerüség szerint redundáns ellenőrző elemekkel látja el.

A kódolással elérhető legegyszerübb hibakorlátozási eredmény a hiba felismerésének képessége.

A hibának csupán a felismerésére alkalmas kódokat *hibajelző kódoknak* nevezzük.

Ha a meghibásodott kódszavakból bizonyos fajta hibák helyére is egyértelmüen következtetni lehet, akkor *hibajavitó kódról* beszélünk. A hibajavitó kódok alkalmasak a javitható hibákon tulmenő, nem javitható hibák jelzésére is.

A gyakorlatban hibajavitásra főleg az ugynevezett lineáris és szisztematikus kódok terjedtek el. A kód akkor lineáris, ha az ellenőrző elemek az adat kódelemeinek lineáris kombinációi, és akkor szisztematikus, ha az ellenőrző elemek nem keverednek az adat kódelemeivel, hanem ezek után külön csoportot alkotnak.

1) A *paritás ellenőrzés* a legegyszerübb hibakorlátozás. A paritásbit páratlan számu hiba jelzésére alkalmas. Sajnos, a hibacsomókban nagy valószinüséggel előfordulnak dupla hibák is, ezért a karakterenként alkalmazott vertikális paritásbit inkább csak szöveges információk közlésénél nyujt elegendő védelmet.

2) Az *aránykódok* valamivel nagyobb redundanciájuak, és igy nagyobb hibajelző képességgel is rendelkeznek. Aránykódnak azokat a kódokat nevezzük, amelyeknek minden kódszava azonos számu "1"-est (s ezért azonos számu "0"-t is) tartalmaz. A leggyakrabban használt aránykód a "4 a 8-ból" kód, amelynek kódszavai 4 darab "1"-est tartalmaznak. A "k" az "m"-ből" kód összes kombinációinak száma:

$$K(k\,;\,m) = \binom{m}{k} = \frac{m!}{m!\;(m-k)!}$$

igy a "4 a 8-ból" kódnak 70 kódszava van.

Az aránykódok az ugynevezett transzpoziciós hibákon - amikor is ugyanannyi "1"-es változik ellenkezőjére, mint amennyi "0" - kivül minden hibát jeleznek. A transzpoziciós hibák valószinüsége azonban szerencsére kicsi.

Az aránykódokat igen ritkán, speciális készülékekben alkalmazzák.

3) A *két dimenziós paritásellenőrzést* már hibajavitásra is fel lehet használni. Ha a karakterek kódjait téglalap alakba rendezzük, akkor vertikálisan (VRC) és longitudinálisan (LRC) is képezhetünk paritásbiteket, sőt az ellenőrző kódszó is ellenőrizhető paritásra:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VRC bitek | | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| | B | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Adat-bitek poziciói | A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| | 4 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| | 2 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| KARAKTEREK | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | X | Y | Z | LRC |

Ezzel a módszerrel magában álló hibás bit javitható. Egyszerüen a vertikális és longitudinális paritásellenőrzés által kijelölt oszlop és sor metszéspontjában található bitet ellentétesre kell változtatni. Rendszerint az aránykódok hibakorlátozó képességének fokozására is felhasználják a longitudinális paritásbiteket.

A két dimenziós paritásellenőrzés jól alkalmazkodik a változó hosszuságu üzenetekhez.

Gyakorlati rendszerekben a két dimenziós paritásellenőrzéssel a felderitetlen hibák száma 100 és 10 000 közé eső faktorral csökkenthető!

4) *A ciklikus kódok*

A gyakorlati adatátviteli rendszerekben használt vonalak tulnyomó többségének tulajdonsága, hogy a hibáknak csak egy része tekinthető véletlennek, viszonylag nagy százalékuk hibacsomó formájában jelentkezik.

A fenti hibakorlátozó kódok használatakor a hibacsomók jelentkezése nagyon megnöveli a nem jelzett hibák valószinüségét. Nagy redundanciáju hibajavitó kódok használata azért nem célszerü, mert ezek azonos hatékonysággal javitják a csomós és többszörös, nem csomós hibákat, holott az utóbbiak sokkal kisebb valószinüségüek.

Ha a "redundanciát ugy épitjük be", hogy elsősorban hibacsomók jelzésére és javitására irányuljon, akkor azonos redundanciával az összes hiba nagyobb hányadát tudjuk felismerni.

Természetesen az ilyen szerkesztésü kódokkal a magányos véletlen hibák is jól javithatók.

A *gyakorlatban legjobban elterjedt*, véletlen és csomós hibákat is hatékonyan jelző és hibacsomókat is javitó kódok a ciklikus kódok.[1] Elterjedtségüket annak is köszönhetik, hogy előállitásuk és dekódolásuk viszonylag egyszerüen, antivalencia kapukon át visszacsatolt léptető regiszter segitségével lehetséges.

A ciklikus kódok lényege a következő:

Az átvinni kivánt blokk bitjeit egy polinom,B(X) együtthatóiként fogjuk fel. A legnagyobb helyértékü bit a legmagasabb fokszámu tag együtthatója.

Átvitel során minden blokkhoz hozzárendeljük még a G(X) generátorpolinomot. G(X) fokszáma nagyobb "0"-nál, de alacsonyabb B(X) fokszámánál, és G(X)-ben $X^0$ együtthatója 1. A két polinom között a moduló 2 algebra szabályai szerint müveleteket végzünk. A müveletek eredményeként az eredeti blokk mögött ellenőrző bitek (BC = block check) keletkeznek, ezeket visszük át utoljára. Az átvitt üzenetből - a generátorpolinom ismeretében - lehet következtetni a hibák nagy részének jelenlétére és egy részének helyére, ez utóbbiak tehát javithatók. A ciklikus kódok jól alkalmazkodnak a változó hosszuságu üzenetekhez.

Hatékonyságuk a generátorpolinom megválasztásától, a hibafajták eloszlásától és az üzenetek hosszától függ.

A blokk végén keletkezett *karakter méretü ellenőrző bitsorozat* közös neve - akár LRC, akár BC jellegü - *blokk ellenőrző karakter*. (BCC - Block check character.)

A *hibakorlátozási stratégiát* az adatátviteli rendszerek a következők közül választják:

a) *Üzenet-visszacsatolás* (loop checking, echo checking)

Ez a módszer nem igényel hibakorlátozó kódolást.

Az adatforrás oldalon tároljuk és az adatvevő oldalon is tárolóba gyüjtjük az üzenet elemeit. A fogadó állomás visszaküldi az egész üzenetet, és ezt az adatforrás oldalon bitről-bitre összehasonlitjuk az eredetivel. A hibásnak talált üzenetet a forrás ismét a csatornára adja. A folyamat előre meghatározható maximális esetben ismétlődhet. Ha a visszacsatolt adatokat az ellenőrzés hibátlannak találja, akkor a forrás egyetlen karakterrel utasitja a vevőoldali tárolót, hogy a blokkot a végleges rendeltetési helyére küldje. Ha a visszacsatolt üzenetet a már elkészitett adathordozóról (lyukszalag, lyukkártya) nyerjük, ezzel kiterjesztettük a védelmet az adatrögzitési szakaszra is. Ekkor gondoskodni kell a hibás adathordozó - lehetőleg automatikus - törléséről vagy kiiktatásáról. A védelem kiterjesztése természetesen még tovább fokozza a csatornakapacitás rossz kihasználását.

---

[1] A ciklikus kódolást a mozgó mágneses felülettel dolgozó háttértáraknál is alkalmazzák hibakorlátozásra.

Látható, hogy az üzenet-visszacsatolás esetében mindkét irányban azonos minőségü adatátviteli rendszer használata célszerü. Alkalmazása duplex vonal esetén előnyös, mivel ilyenkor a visszaut rendszerint ugyis kihasználatlan lenne. Ezt a stratégiát elterjedten használják telephelyi rendszerek viszonylag rövid átviteli vonalainál, ahol az egyébként is alacsony kihasználatlanságu vonalakon ez nem okoz lényeges költségtöbbletet.

b) *Automatikus ismétlés kérés*
(Automatic repetition request; ARQ)

Ennél az eljárásnál általában hibajelző kódolót és dekódolót alkalmazunk. A dekódoló által jelzett hibákról a vevőoldal automatikusan hibajelentést küld vissza az adónak, mire az megismétli a hibásnak itélt üzenetet.

Bár kétirányu átvitelre itt is szükség van, a visszacsatoló csatorna minősége jóval az adó csatornáé alatt maradhat.

Ennél az eljárásnál az alkalmazott hibakorlátozó kód, az üzenetek hossza, a vonalak tulajdonságai és az adás-ismétlés automatizáltságának foka viszonylag szabadon összehangolhatók, ezért *ez a stratégia terjedt el a legjobban* a gyakorlatban.

Az ismétlések száma lényegesen csökkenthető, ha *hibajavitó kódolást* alkalmazunk, és csak a nem javitható hibák esetén kér a vevőállomás ismétlést. Ez a javitott ARQ módszer az integrált áramkörökkel olcsón realizált hibajavitó áramkörök alkalmazásának köszönhetően várhatóan széles körben el fog terjedni.

Az ARQ rendszerek többségében az adóállomás a helyes vételről *nyugtázást* (ACK) igényel, csak kevés rendszer tekinti a *nemleges válasz* (NAK) hiányát hibátlan átvitelnek (ld. 6. fejezet).

Ha egy centrális forrás (Master) egyszerre több vevőnek ad, akkor egyetlen csatorna hibája miatt az ismétlés ideje alatt az összes többi csatornák tétlenek lennének, ezért ilyen esetekben általában ez a módszer nem felel meg. Néha időazonos rendszerekben sem engedhető meg a gyakori ismétlésekkel járó időveszteség, ilyen esetekben a következő stratégiát használják.

c) *Közvetlen hibajavitás* (Forward error correction: FEC)

Az eljárás hatékony hibajavitó kódot igényel. A hibakorlátozó dekódoló igen bonyolult és költséges, ha az adatvonalak nagy hibaaránya vagy a hosszu hibacsomók miatt erősen redundáns hibajavitó kódot használunk. Ez a stratégia nem igényel visszacsatoló csatornát. Nagysebességü egyirányu összeköttetések esetén alkalmazása gazdaságos lehet. Csak speciális rendszerekben alkalmazzák.

Egy távadatfeldolgozó rendszer tervezésénél fontos ismernünk az egész rendszer biztonságára vonatkozó elvárásokat, a teljes rendszer müködési szüneteinek, illetve a rendszerelemek hibás müködésének hatását a megoldandó feladatokra.

Ennek a bonyolult problémának csak része a zajos csatornán történő adatátvitel védelme.

Ma a távadatfeldolgozó rendszerek hibavédelmi eljárásai a komplex hibavédelem irányában fejlődnek. Ez azt jelenti, hogy logikailag egyre hosszabb szakaszokat vonunk be az egységes védelmi eljárásba, kezdve az adatok első rögzitésétől és beolvasásától a közbenső állomásokon át a feldolgozásig.

A védelem kiterjeszthetősége azon alapul, hogy minden védelmi eljárás redundanciát igényel. Komplex hibavédelem esetén a rendszerbe vitt redundancia a rendszer egy ellenőrzési pontján történő kiértékelés során nem vész el, hanem tovább halad egy következő ellenőrzési pont felé, vagyis több átviteli és feldolgozási fázison is áthaladhat.

Fontos kiemelni, hogy a távadatfeldolgozó rendszerhez kapcsolódó ember általában két-, három nagyságrenddel többet téveszt, mint a rendszer legmegbizhatatlanabb elemei. Ezért rendkivül lényeges az előkészitő emberi munka és az ember-gép-illesztés szigoru ellenőrzése.

# 6.

# Az adatátvitel vezérlési eljárásai

A távadatfeldolgozó rendszer adatállomásai közötti kommunikáció a helyi feldolgozásnál megszokott periféria input-outputhoz képest merőben uj eljárásokat igényel. Az ujszerüség alapvető oka az, hogy az adatállomások között (általában) csupán egyetlen érpár áll rendelkezésünkre a vezérlés lebonyolitására és a hasznos információ átvitelére. (Csak az általánosnak tekinthető soros átvitellel foglalkozunk, ezen belül is elsősorban a félduplex vonalak vezérlésével, ez ugyanis a duplex vonali vezérlés minden lényeges elemét tartalmazza.) A sorosan érkező, különböző jelentésü vezérlő karakterek és a hasznos információ szétválaszthatóságának előfeltétele, hogy az *üzenetek* előzetes konvenciók alapján, *szigoru protokoll szerint* épüljenek fel. A vezérlést tovább bonyolitja, hogy az adatállomások nagy része nem érheti el egymást bármely pillanatban közvetlenül, hanem csak valamilyen kapcsolt hálózat segitségével átmenetileg létrehozott összeköttetés utján, és hogy egy közös adatvonalon több felfüzött adatállomás is osztozkodhat.

Ugyancsak a vezérlés részét képezi az információcserét lebonyolitó adatállomások szinkronizmusának létrehozása és az adatátviteli hibák kezelése.

A vezérlési eljárások közös jellemzője, hogy a tényleges üzenettovábbitást *vezérlő karakterekkel folytatott "párbeszéd"* (handshaking) előzi meg, illetve kiséri.

Az egymással kommunikáló adatállomások a vezérlés szempontjából rendszerint nem egyenranguak, az irányitás a vezérállomás, a központi számitógép feladata, és a vezérlési eljárás a hardware és a software összehangolt tevékenységét igényli. Ugyanakkor az eljárás bonyolultságától függően bizonyos "intelligenciát", párbeszéd lebonyolitó képességet az alállomási szinten is fel kell téteieznünk, akkor is, ha az alállomás egyszerü adatvégállomás.

Számitógépes adatállomások kapcsolata esetén a vezérállomási feladatkör általában dinamikusan váltakozva hozzárendelhető bármelyik adatállomáshoz.

Az adatátvitel vezérlési eljárása igen sok tényezőtől függ: befolyásolja az adatvonali és az összeköttetési üzemmód, az összeköttetési forma, a továbbitott információ felbontásának módszere, az alkalmazott adatátviteli technika, a forráskód, a hibakorlátozó kód és a hibakorlátozási stratégia, az összekapcsolt adatállomások vezérlő és vezérlés feldolgozó képessége, intelligenciájának foka stb. Mivel a fenti tényezők egyetlen rendszeren belül is igen heterogén halmazt képező kombinációkban fordulhatnak elő, általában szükség van többféle adatátviteli vezérlési eljárás párhuzamos alkalmazására.

Ha a központi számitógép el van látva megfelelő adatátviteli vezérlő programcsomaggal, akkor a rendszer felhasználójának nem kell pontosan ismernie az egyes fizikai vonalakra érvényes adatátviteli vezérlési eljárások minden apró részletét. Ezeket az információkat a rendszer generálásakor beépitjük a rendszerbe, igy implicit formában később rendelkezésünkre állnak. A felhasználónak azonban szüksége van az adatátviteli vezérlési eljárások általános ismeretére és az adatátviteli vezérlő karakterek használatára, hogy programjának üzenetkezelő részét megfelelő jelentésü és sorrendü vezérlő információkkal láthassa el.

A létező sokféle vezérlési eljárást több szempont szerint csoportosithatjuk. Mindenekelőtt meg kell különböztetnünk a vonalkapcsolás nélküli és vonalkapcsolást igénylő összeköttetések vezérlését.

*Vonalkapcsolás nélküli esetben* is szükség van a partnerek közötti kapcsolatteremtésre, mivel sem a közvetlen, sem a felfüzéses összeköttetés nem jelent állandóan aktiv vonalat.

A legegyszerübb esetben a vonal két végén elhelyezett két adatállomás áll egymással szemben (pont-pont összeköttetés), amelyek *"vetélkednek"* a vonal használatáért (contention).

Amelyik adatállomás üzenetet kiván küldeni, az egy speciális karakterrel felhivja partnere figyelmét. A speciális karaktert kibocsátó adatállomás *üzenete végéig birtokolja* a vonalat. Az üzenet végét egy másik speciális karakter jelzi, amely alapállapotba hozza a vonalat, hogy bármelyik adatállomás megkaphassa a vezérlést. Ha mindkét adatállomás egyszerre igényli a vonalat, akkor a vezérállomás kapja meg. Ezt ugy érjük el, hogy a vezérállomás "türelmi ideje" (time out) rövidebb, ezért kérését előbb ismételheti meg.

A *felfüzéses hálózatban* dolgozó adatállomások is használhatják közös adatvonalukat a fenti, *igény alapon történő szervezés* egy továbbfejlesztett változata segitségével, ezt azonban ritkán alkalmazzák. Annál elterjedtebb a felfüzött adatállomások adatátvitelének vezérlésére a már röviden emlitett polling-selecting technika. Mielőtt ezt a nagyjelentőségü eljárást részletesen megvizsgálnánk, tekintsük át röviden a kapcsolt hálózatokban felmerülő vezérlési problémákat.

A vonalkapcsolt hálózat elsősorban pont-pont összeköttetések létrehozását szolgálja, de a felkapcsolt vonalszakaszon lehetnek felfüzött adatállomások is. Ez azt jelenti, hogy általában a fenti módszerek bármelyikét alkalmazhatjuk, ha előzőleg a vezérlés részeként létrehoztuk a kapcsolatot.

A kapcsolat létrehozása egy *felhivási eljárás* lebonyolitását jelenti. Általában mind a vezérállomás, mind az alállomás(ok) képes(ek) a felhivás végrehajtására, néha azonban csak az egyik hivhatja a másikat. A számitógépes adatállomások a "tárcsázás" automatikus leutánzásával (autocall), az adatvégállomások többnyire manuális tárcsázással dolgoznak. A felhivott adatállomás rendszerint *automatikusan válaszolja meg a hivást*. A számitógépes adatállomás csak abban az esetben válaszol pozitivan ("veszi fel a kagylót"), ha erre előzőleg a felhasználó programjától - az adott vonalra vonatkozóan *engedélyt* (enable) kapott

Engedélyezés hiányában a hivó alállomás foglalt vonalat lát. Természetesen akkor is foglalt a vonal, ha egy másik adatállomás használja.

Vonalkapcsolásnál előfordulhat, hogy egy alacsony megbizhatóságu vonalszakasz kerül az összeköttetésbe. Ilyenkor általában segit az ujrahivás, ami programmal automatikusan is elvégezhető.

*Üzenetkapcsoló hálózatokban* az üzenetkapcsoló központ és az adatállomások között a kapcsolat vagy közvetlen, vagy felfüzéses. Az üzenetet magát azonban az üzenetkapcsoló központnak tovább kell adni, ezért minden üzenetet irányitó információkkal, fejléccel (header) látunk el. A fejléc tartalmazhatja a rendeltetési hely(ek) cimét (cimeit), a "feladó" cimét, az üzenet azonositóját, prioritását stb.

A felfüzéses hálózatok alállomásai rendszerint nem képesek egymással közvetlen kapcsolatba lépni, hanem csupán a vezérállomás - mint üzenetkapcsoló központ - segitségével kommunikálhatnak.

Mivel az adatkoncentrátorok is az üzenetkapcsolás elvén müködnek, a fentiek igazak az adatkoncentrátorokra is. Természetesen olyan megállapodás tehető, hogy a végállomásoktól fejléc nélkül érkező üzenetek az egyetlen feldolgozó központnak szólnak.

A fentiekből látható, hogy üzenetkapcsolás esetében az adatátvitel vezérlése lényegében megegyezik az üzenetkapcsolás nélküli esettel, a különbség csupán az, hogy az üzenet elején a *szöveg előtt fejlécet* is alkalmazunk.

Az eddigiek összefoglalásaként azt mondhatjuk, hogy lényegében a legfontosabb vezérlési eljárás a polling-selecting technika. A pont-pont összeköttetésnél alkalmazott igény alapon történő vezérlés felfogható ugy, mint ennek speciális esete, a kapcsolt hálózatokban alkalmazható vezérlések pedig ugy, mint ennek bővitései.

A hagyományos adatátviteli vezérlések *karakter orientált eljárások*. Napjainkban csak ezek terjedtek el széles körben.

A sok módszer között mindenek előtt aszerint érdemes különbséget tenni, hogy aszinkron vagy szinkron adatátviteli vezérlési eljárásról van-e szó.

A legismertebb karakter orientált szabványos eljárások az ANSI-CCP (ASCII kódra vonatkozó szabványos amerikai eljárás), az ISO-BMCP[1] (ISO/CCITT 7-elemes kódjára vonatkozó alapeljárás) és az IBM-BSC (az IBM cég bináris szinkron adatátviteli eljárása). A továbbiakban nem valamelyik konkrét eljárás részleteit, hanem csak egy-egy általánosnak tekinthető aszinkron és szinkron eljárás polling-selecting részét tekintjük át. Ez után röviden megvizsgáljuk egy uj, *bit-orientált szinkron eljárás* lényegét.

---

[1] A kézirat javitásakor ennek alapján készül az MSZ 7790-jelü, "Alapmódu vezérlőeljárások adatátviteli rendszerekhez" cimü, a tervezet szerint 1975. jan. 1-én hatályba lépő Magyar Szabvány.

## *6.1. Az aszinkron adatátvitel vezérlése*

Az aszinkron adatátviteli eljárások igen sokfélék, és vezérlő karaktereik elnevezése kevésbé szabványositott, mint a szinkron eljárások vezérlő karaktereié. Az egyszerüség kedvéért vezessük most be a következő vezérlő karakter jelöléseket:

Ⓐ = Átvitel vége
Ⓑ = Blokk vége
Ⓒ = Cim vége
Ⓝ = Negativ válasz
Ⓟ = Pozitiv válasz

Ezeknek a vezérlő karaktereknek vagy egy-egy kódszó, vagy - ha külön vezérlő jelek nincsenek a kódban - két-két, kötött sorrendü kódszó felel meg.

A vezérlő karakterek megfelelő elhelyezése software feladat. Az aszinkron átvitelnél alkalmazott Start és Stop bitek generálását és elhagyását, valamint az átviteli hibák felismerését a hardware végzi.

A vezérállomás központi számitógépéhez kapcsolt adatátviteli vezérlő egység müködés közben a következő állapotokat veheti fel:

vezérlési állapot (control mode),
szöveg-vétel állapot (text-in mode),
szöveg-adás állapot (text-out mode).

Ezekre az állapotokra azért van szükség, hogy az egyes vezérlő karaktereknek több értelmezési lehetősége, illetve jelentése is lehessen.

Képzeljük el, hogy a vezérállomás olyan felfüzéses hálózatot vezérel, amelyhez egynél több terminál egységet kiszolgáló terminál vezérlő egységek csatlakoznak. Igy egy adott terminál kiválasztásakor külön megkell nevezni a terminál egységet is. (Az aszinkron átvitelt csaknem kizárólag számitógép-terminál relációban alkalmazzák.)

Tételezzük fel, hogy *longitudinális paritásellenőrzést* és az *automatikus ismétlés kérés* stratégiáját használjuk. Aszinkron átvitelnél néha egyáltalán nem alkalmaznak hibakorlátozást, de ha igen, akkor általában ezt az eljárást követik.

A 11. ábra mutatja a polling, a 12. ábra a selecting jelfolyamatát.

Először nézzük a lekérdezést!

A központi számitógépből érkező Ⓐ jel hatására mind az adó-, mind a vevőoldali hibakorlátozó logika, valamint az összes terminál *alapállapotba*, az adatátviteli vezérlő egység pedig *vezérlési állapotba* jut. A következő "X" karakter a lekérdezendő terminál vezérlő egység, a "3" pedig az általa vezérelt egyik input terminál egység cime.

Ha a terminál vezérlő egység csak egyetlen terminál egységet vezérelne, akkor elegendő lenne egyetlen karakterből álló terminál cim kiadása is. Ha a megcimzett egység nem kiván üzenetet küldeni, ezt az Ⓝ karakterrel jelzi. Ⓝ vétele után az adatátviteli vezérlő egység vezérlési állapotban marad, tehát folytatódhat a cim jellegü karakterek kiadása, vagyis a lekérdezési ciklus. Most az "Y" terminál vezérlő egység "5"-ös input terminál egységét cimezzük meg. A cim talál, a terminál adásra kész és ezt a Ⓒ karakterrel jelzi vissza. Ⓒ hatására az adatátviteli vezérlő egység szöveg-vétel állapotba jut. A következő karaktertől kezdve (első szövegkarakter) müködésbe lép a hibakorlátozó logika, mind az adó, mind a vevő oldalon elkezdődik a longitudinális ellenőrző karakter generálásához szükséges jelek akkumulációja. A szövegvétel állapotban a szövegben előforduló Ⓒ, Ⓝ és Ⓟ vezérlő karakterek hatástalanok, egyszerüen a szöveg részeként jelennek meg. A szöveget az üzenetküldő terminál a Ⓑ jellel zárja le. Ez azt jelenti, hogy a következő karakter az adó által akkumulált ellenőrző karakter (LRC) lesz. A vevő ezt összehasonlitja az általa generált LRC-vel, és egyezés esetén Ⓟ jellel nyugtázza a vételt. Ha az adatátviteli vezérlő egység hibát észlelt, Ⓝ jellel ismétlést kér. A terminál Ⓟ -re Ⓐ -val válaszol, ez az adatátvitel végét jelenti, az adatátviteli vezérlő egység vezérlési állapotba kerül.

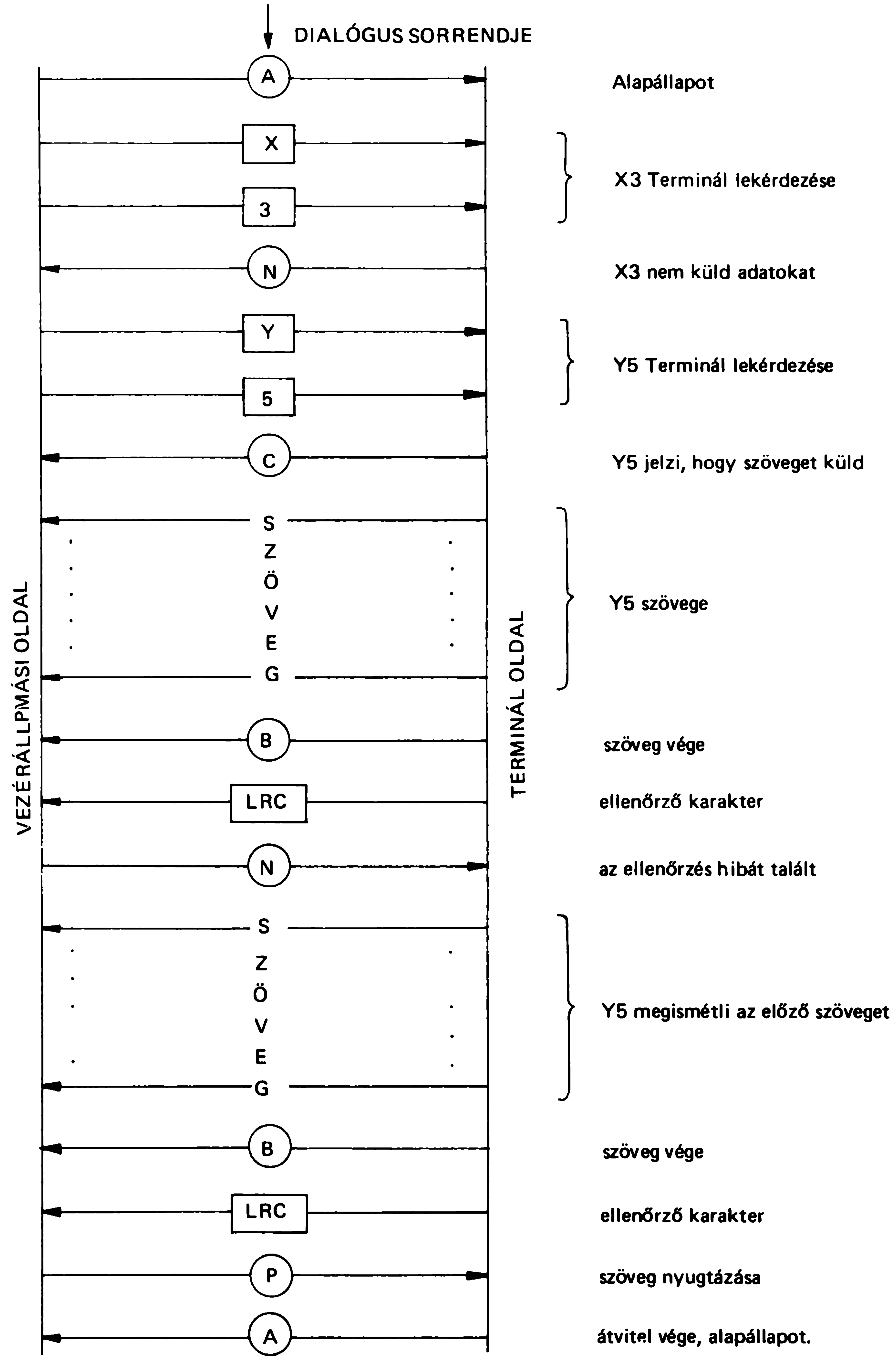
A kiválasztás folyamata (12. ábra) a fentiek alapján már könnyen érthető. Talán annyit elég az ábrához fűzni, hogy Y6 lehet a lekérdezésnél Y5-nek nevezett terminál egység *output esetében érvényes cime is*, de jelenthet egy másik output terminál egységet is.

Pont-pont összeköttetés esetében a fenti procedura igen leegyszerüsödik: minden üzenet egyszerüen Ⓒ-vel kezdődik és Ⓐ -val végződik, előzetes kiválasztás nincs, minden üzenet egy "közvetlen kiválasztás".

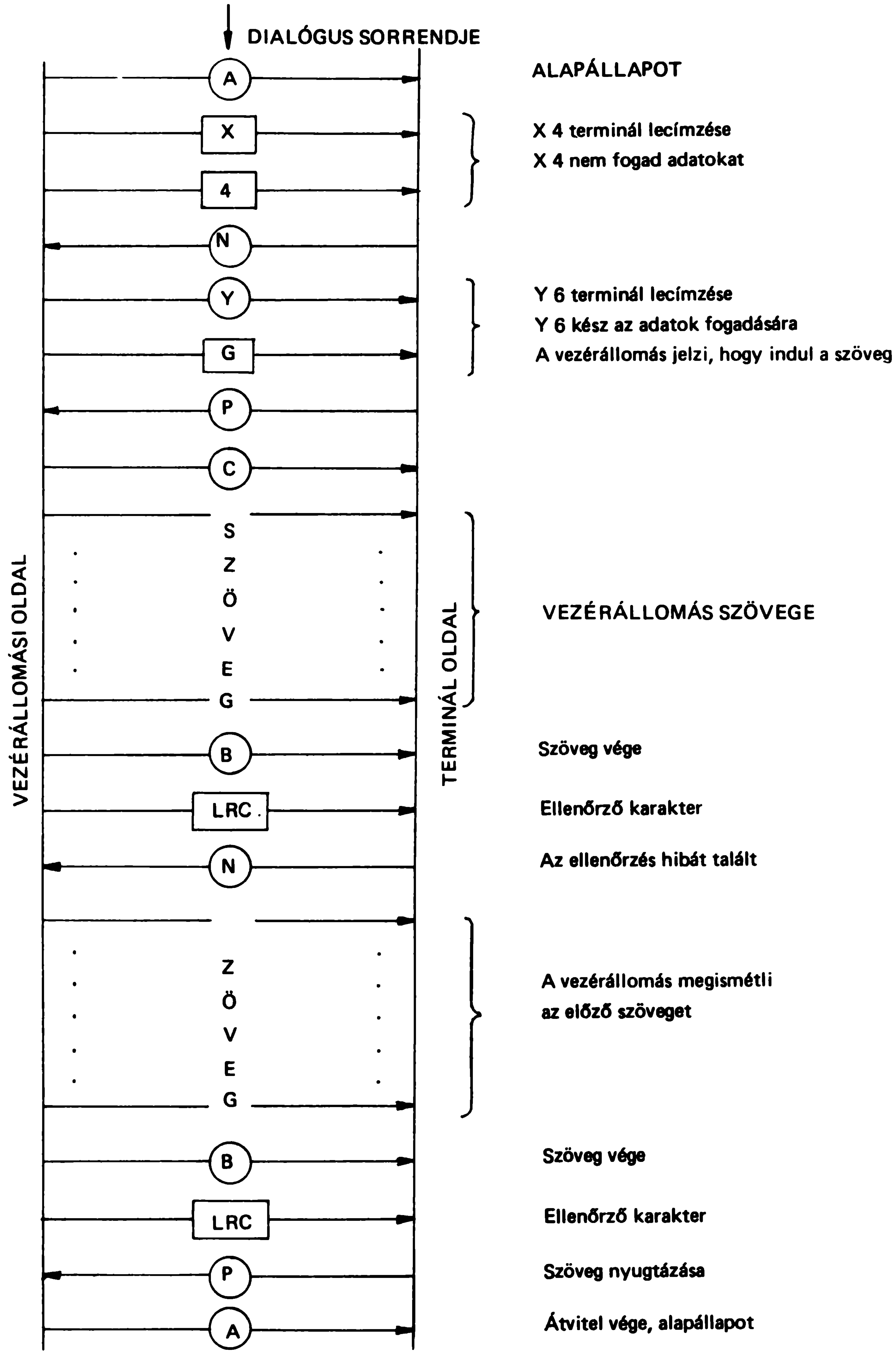
Ⓒ azt jelenti, hogy a következő karakter szöveg, A pedig alapállapotba hozza a rendszert. Mivel bármelyik adatállomás kezdeményezhet, ez tulajdonképpen "vetélkedés".

## *6.2. A szinkron adatátvitel vezérlése*

A karakter orientált szinkron adatátvitel vezérlésének alapelve szintén az egymással kommunikáló adatállomások között, vezérlő karakterekkel folytatott "párbeszéd" (handshaking).


DIALÓGUS SORRENDJE
A
Alapállapot
X
3
X3 Terminál lekérdezése
N
X3 nem küld adatokat
Y
5
Y5 Terminál lekérdezése
C
Y5 jelzi, hogy szöveget küld
SZÖVEG
Y5 szövege
VEZÉRÁLLPMÁSI OLDAL
TERMINÁL OLDAL
B
szöveg vége
LRC
ellenőrző karakter
N
az ellenőrzés hibát talált
SZÖVEG
Y5 megismétli az előző szöveget
B
szöveg vége
LRC
ellenőrző karakter
P
szöveg nyugtázása
A
átvitel vége, alapállapot.

*11. ábra*
*Lekérdezés aszinkron adatátvitelnél*


DIALÓGUS SORRENDJE
VEZÉRÁLLOMÁSI OLDAL
TERMINÁL OLDAL
A
X
4
N
Y
G
P
C
SZÖVEG
B
LRC
N
ZÖVEG
B
LRC
P
A
ALAPÁLLAPOT
X 4 terminál lecímzése
X 4 nem fogad adatokat
Y 6 terminál lecímzése
Y 6 kész az adatok fogadására
A vezérállomás jelzi, hogy indul a szöveg
VEZÉRÁLLOMÁS SZÖVEGE
Szöveg vége
Ellenőrző karakter
Az ellenőrzés hibát talált
A vezérállomás megismétli az előző szöveget
Szöveg vége
Ellenőrző karakter
Szöveg nyugtázása
Átvitel vége, alapállapot

*12. ábra*
*Kiválasztás aszinkron adatátvitelnél*

A szinkron adatátviteli eljárásokat azonban nagyobb sebesség, hatékonyabb hibakorlátozás és több, programmal választható járulékos szolgáltatás, a vezérlési folyamat módosithatósága jellemzi.

A szinkron átvitelt mindenekelőtt "intelligens" adatállomások viszonylatában alkalmazzák, de számos közepes-, vagy nagysebességü adatvégállomás is szinkron átvitellel dolgozik.

A vezérállomás adatátviteli vezérlő egysége *adás* (transmit) és *vétel* (receive) *üzemmódra* képes. Mindkét üzemmódban felveheti a *vezérlési* (control) és a *szöveg* (text) *állapotát.* A szöveg átvitele történhet *normál-* és *transzparens,* "átlátszó" (transparent), vagyis kódszó-független módon.

Az adatállomások szinkronizmusáról és az átviteli hibák felismeréséről a hardware gondoskodik, a vezérlő karakterek megfelelő elhelyezése és fogadása azonban vezérállomási sfotware feladat.

Megint képzeljük el az aszinkron átvitel vezérlésének vizsgálatánál feltételezetthez hasonló felfüzéses hálózatot.

A normál módra jellemző lekérdezési folyamat a 13. ábrán követhető végig.

Az egyszerüség kedvéért az egyes "megszólalásokat" bevezető 2-5 karakterből álló szinkronjel-sorozatot Ø-vel jelöltük.

Az ábra három esetet mutat be:

1.) A lekérdezett adatállomás (X3) nem kiván üzenetet küldeni
2.) A lekérdezett adatállomás (Y4) üzenete egyetlen blokkból áll.
3.) A lekérdezett adatállomás (Z5) üzenete "m" blokkból áll.

A "párbeszédet" az előző fejezetben a kódoknál felsorolt adatátviteli vezérlő karakterek segitségével végzik az adatállomások.

Tételezzük még fel, hogy az üzenet hibakorlátozását ciklikus kódolással végezzük ARQ stratégia szerint. Az alkalmazott generátorpolinom a $G(X) = CRC_{16} = X^{16} + X^{2} + 1$ polinom, amely két nyolcbites ellenőrző karaktert (BCC) állit elő. Ezt a generátorpolinomot igen elterjedten alkalmazzák a 8-bites kódstrukturák (EBCDIC, ASCII-8) hibakorlátozásánál.

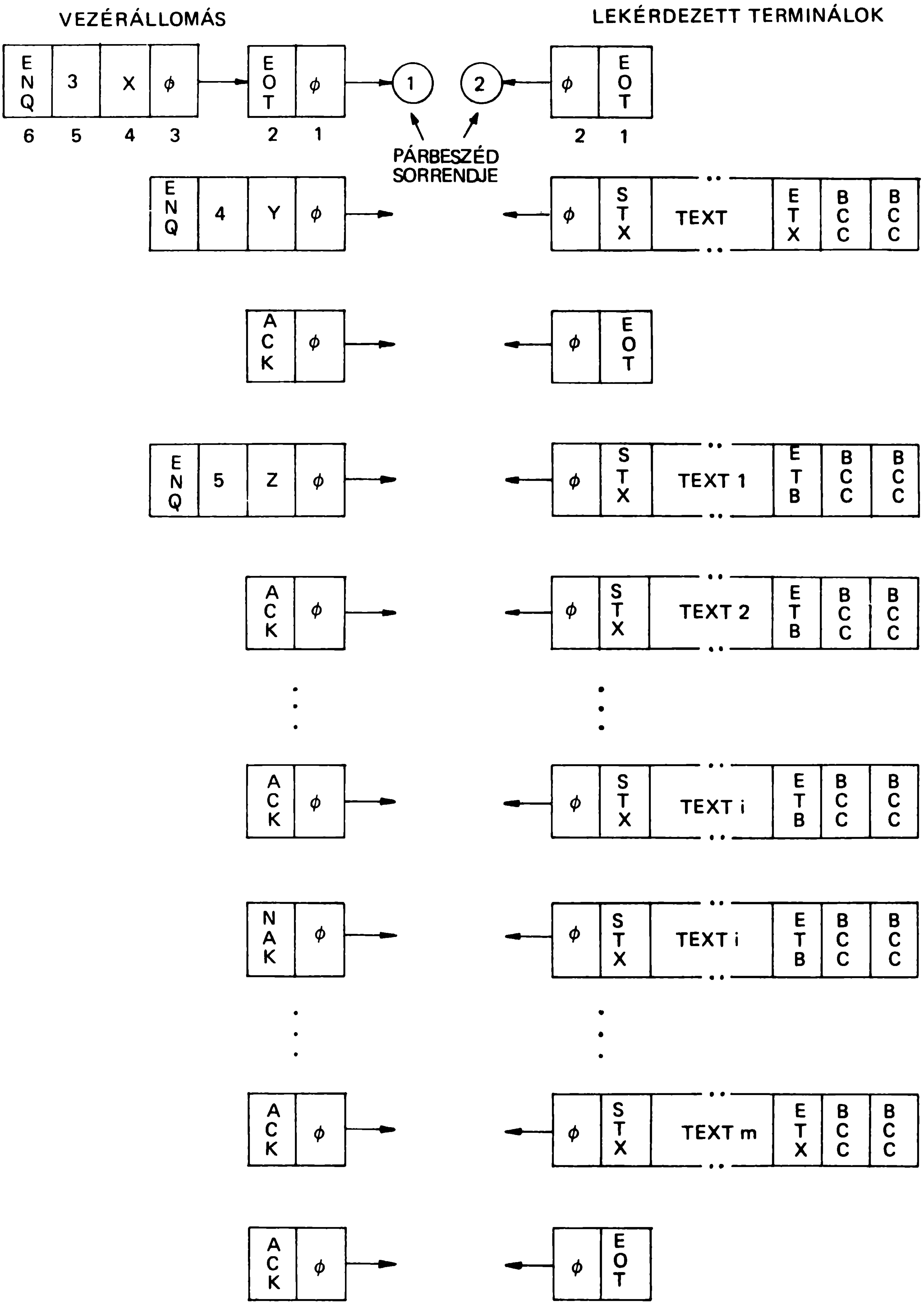
A lekérdezési ciklus előtt a vezérállomás *EOT, Adás vége* jellel alapállapotba hozza az átviteli rendszert. Az adatátviteli vezérlő egység *adás üzemmódba* és *vezérlési állapotba* kerül.

A vezérállomás a vonalra adja a lekérdezendő terminál vezérlő egység és terminál egység cimét, X3-at, s a cimrészt *ENQ, Kérdezés* jellel zárja. Az ENQ egyben válaszra szólitja fel a lekérdezett terminált, hatására az adatátviteli vezérlő egység vétel üzemmódra vált át.

A kérdésre X3 EOT-vel negativ választ ad, visszaadja a vezérlést az adatátviteli vezérlő egységnek.

A lekérdezett Y4 üzenetét *STX, Szövegkezdés* jellel inditja, és *ETX, Szöveg vége* jellel zárja. Üzenetkapcsolás esetében az STX helyén áll az *SOH Fejléckezdet,* és a fejléc végén következik az STX jel.

Néha a fejléc átvitelére külön blokkot használnak, ilyen esetben a fejléc ugy fejezhető be, mintha szöveg blokk lenne.


VEZÉRÁLLOMÁS
LEKÉRDEZETT TERMINÁLOK
ENQ
3
X
ϕ
6 5 4 3
EOT
ϕ
2 1
1
2
PÁRBESZÉD SORRENDJE
ENQ 4 Y ϕ
ϕ STX TEXT ETX BCC BCC
ACK ϕ
ϕ EOT
ENQ 5 Z ϕ
ϕ STX TEXT 1 ETB BCC BCC
ϕ STX TEXT 2 ETB BCC BCC
ϕ STX TEXT i ETB BCC BCC
NAK ϕ
ϕ STX TEXT m ETX BCC BCC

*13. ábra*
*Lekérdezési folyamat szinkron adatátvitelnél*

STX vagy SOH hatására az adatátviteli vezérlő egység *szöveg állapotba* kerül és megkezdődik a hibakorlátozó logikában az ellenőrző karakterek akkumulációja.

Az ETX visszaállitja a *vezérlési állapotot.*

Az ETX-et két BCC, ellenőrző karakter követi.

A vezérállomás a válaszadás idejére *adás üzemmódra* vált, majd ACK jellel nyugtázza a vételt, és ujból *vétel üzemmódba* kerül.

A terminál EOT-vel visszaadja a vezérlést a vezérállomásnak.

A Z5 terminál üzenete több blokkból áll, ezért a közbülső szövegrészek végét ETX helyett *ETB Átviteli blokk vége* jel zárja.

Az utolsó blokk ETX-szel fejeződik be, amit nyugtázás után EOT követ. Ha a vezérállomás valamelyik blokkot hibásnak találta, akkor ACK helyett NAK-kal válaszol, és ezzel ismétlést kér.

A fentiek alapján a 14. ábrán látható kiválasztási folyamat értelemszerüen végigkövethető.

Ha a kiválasztott adatállomás nem kész a vételre, akkor ezt NAK jellel jelzi vissza. Ugyancsak NAK-kal jelzi a hibás vételt.

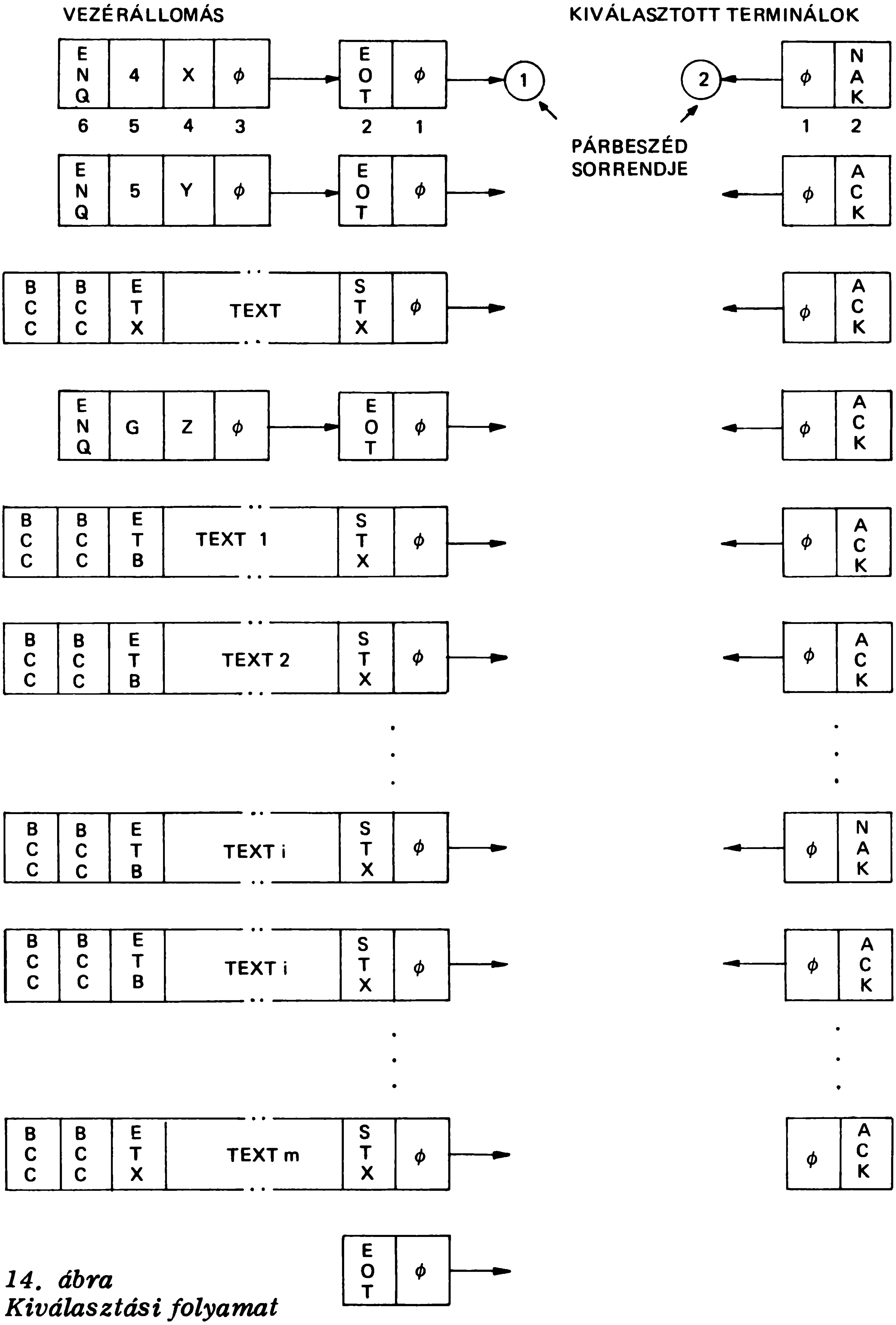
A gyakorlatban - lekérdezésnél és kiválasztásnál egyaránt - az egymást követő üzenet-blokkokat felváltva $ACK_0$ és $ACK_1$ jelekkel nyugtázza a vevőállomás. Igy az adóállomás "észreveheti", ha egy teljes adatblokkja elveszik a vonalon.

Pont-pont összeköttetés esetén a vezérlési eljárás lényegében a kiválasztással egyezik meg, bármelyik adatállomás kezdeményezhet, de nyilván az ENQ előtt nincs szükség cim megadására.

A szinkron eljárásoknál a vezérlési folyamat általában flexibilisen módositható. Különleges vezérlő karakterek segitségével például a vezérállomás kis ideig késleltetheti az adást, a terminál is felkérheti a vezérállomást az adás átmeneti késleltetésére, a terminál megszakithatja a vezérállomás üzenetfolyamát, hogy sürgős üzenetét a vonalra adhassa. Létezik olyan ARQ megoldás is, amely pozitiv választ (ACK) nem igényel, hanem amennyiben egy "türelmi időn" belül nem kap negativ választ (NAK), akkor hibátlannak fogadja el az átvitelt. Ezzel az effektiv átviteli sebesség jelentősen gyorsitható, mivel sok irányváltási periódust lehet megtakaritani. Ezeket a megoldásokat most nem részletezzük.

Ehelyett röviden nézzük meg, hogyan biztositható az "átlátszó" szöveg átvitel. Ilyen kódszó-független átvitelt igényel például a tömöritett (packed) decimális információnak két számitógép közötti átvitele, forrás-, illetve tárgyprogramok továbbitása, általában az operativ tár egy részének átmásolása egy másik operativ tárba, illetve tetszőleges bináris adatcsoportok átvitele. A transzparens átvitel lehetővé teszi, hogy a szövegben az adott kód bármely kódszava, - tehát a vezérlő jelek is -, tetszőleges kombinációban előforduljon.

A transzparens átvitel lényegében megegyezik a fenti lekérdezési-kiválasztási eljárással. A különbség az, hogy azokat a vezérlő karaktereket, amelyektől vezérlő hatást várunk, megelőzi egy *DLE*,


VEZÉRÁLLOMÁS
KIVÁLASZTOTT TERMINÁLOK
ENQ 4 X ϕ
6 5 4 3
EOT ϕ
2 1
1
2
ϕ NAK
1 2
PÁRBESZÉD SORRENDJE
ENQ 5 Y ϕ
EOT ϕ
ϕ ACK
BCC BCC ETX TEXT STX ϕ
ϕ ACK
ENQ G Z ϕ
EOT ϕ
ϕ ACK
BCC BCC ETB TEXT 1 STX ϕ
ϕ ACK
BCC BCC ETB TEXT 2 STX ϕ
ϕ ACK
BCC BCC ETB TEXT i STX ϕ
ϕ NAK
BCC BCC ETB TEXT i STX ϕ
ϕ ACK
BCC BCC ETX TEXT m STX ϕ
ϕ ACK
EOT ϕ

*14. ábra*
*Kiválasztási folyamat szinkron adatátvitelnél*

*Adatátviteli-átkapcsolás* jel. Ha a szöveg DLE, STX-szel kezdődik, akkor az adatátvitel "átlátszó" lesz. Ha az átviendő karakterek között DLE-nek megfelelő bitkombináció lép fel, akkor két DLE kerül közvetlenül egymás után átvitelre, melyek közül az egyiket a vevőállomás automatikusan elhagyja. A transzparens átvitel addig tart, amig DLE, ETB; vagy DLE, ETX-szel vissza nem térünk a normál müködésre.

## *6.3. Bit-orientált szinkron adatátvitel*

A karakter-orientált adatátvitel vezérlési eljárásokat a 60-as évek elején, elsősorban a kötegelt feldolgozás adatátviteli szükségleteinek kielégitésére fejlesztették ki. Később az uj, rövid üzenetekkel dolgozó interaktiv alkalmazásokhoz szinte cégenként különböző változtatásokat, kiegészitéseket végeztek, igy sok inkompatibilis eljárás-csoport alakult ki, amelyek megnehezitik az adaptációt, és a nagy rendszerek hatékony kiépitését. A heterogenitás mellett a karakter-orientált eljárások közös hátrányai a következők:

1.) Minden eljárásnál előzetes és kisérő párbeszédre van szükség, ami rövid üzeneteknél nagyon rossz átviteli hatásfokot eredményez.

2.) Az eljárások előirják a vezérlő és szöveg-információ azonos kódstrukturáját (pl. 8-bites kódszavak).

3.) Az átvitt információ felbontási lehetőségei korlátozottak.

4.) Sokféle, különböző hatékonyságu hibakorlátozási eljárást alkalmaznak, de a vezérlő karakterekre általában csak egyszerü paritásvédelem vonatkozik, az üzenet ellenőrzésének mértéke nem egyenletes.

Ezeket a korlátozó tényezőket hivatott a bit-orientált vezérlési eljárás csökkenteni, illetve kiküszöbölni. (Ezt az uj szinkron adatátviteli eljárást az IBM javasolta 1969-ben SDLC eljárás néven, majd az ANSI és az ISO továbbfejlesztve, gyakorlatilag azonos módon szabványositotta, rendre ADCCP, illetve HDLC elnevezéssel.)

Egyes vélemények szerint várható, hogy ez az eljárás a hagyományos szinkron átviteli eljárások helyébe lép.[1]

Az eljárás vonalkapcsolt és vonalkapcsolás nélküli, félduplex és duplex vonalakon, közvetlen vagy felfüzéses összeköttetésnél, "intelligens" adatállomások közötti és terminál-számitógép relációban egyaránt alkalmazható.

Az eljárás a 15. ábrán látható adatátviteli információ egységre, *Keret-strukturára* (Frame) épül.

A Keretet nyolc-bites *Zászlók* (Flag) határolják. A Zászlót mindig a 01111110 bitkombináció jelenti.

---

[1] SZÁMOK, Távadatfeldolgozás c. speciális tanfolyam, Balatonkenese, 1973.

| | | | | | |
|---|---|---|---|---|---|
| FLAG | ADDRESS | CONTROL | INFORMATION | FCS | FLAG |

*15. ábra*
*Bit-orientált átvitel keret-szerkezete*[1]

A kezdő Zászlót a *Cim* (Address) követi, amely után *Vezérlés* (Control) információ következik. A Zászló, Cim és Vezérlés együtt alkotják a nyitó *Összekapcsolás vezérlést* (Link control).

Ezt követi a *Szöveg*, amelyet *Hálózatvezérlő információk* előzhetnek meg. Ezek képezik az *Információs mezőt* (Information).

A Szöveget a záró *Összekapcsolás vezérlés* követi, amely *Keret-ellenőrző részből* (FCS) és *Zászlóból* (Flag) áll.

A nyitó és záró Összekapcsolás vezérlés 8 bit egész számu többszöröse. Rendszerint a Zászlók, a Cim és a Vezérlés 8 bit hosszuságuak, az FCS pedig 16 bites, de az egyes mezők összekapcsolással bővithetők.

Ugyanakkor az Információs mező tetszőleges bithosszuságu lehet. Hardware gondoskodik arról, hogy a Zászlókon belüli részben az átvitel során soha ne forduljon elő ötnél több egymást követő "1"-es. Ha ilyen fellépne, ezt adásnál automatikus "0" beillesztéssel korrigáljuk, s az ilyen "0"-ákat a vevő automatikusan kitörli. Az adatállomások szinkronizmusáról - itt nem részletezett módon - automatikusan gondoskodik a rendszer.

Az eljárás fontos koncepciója, hogy a vezérlésért egy rögzitett adatállomás, a *Primer állomás* felel, az összes többi, ugynevezett *Szekunder adatállomás* csak a Primer állomás *Parancsaira* (Command) *Válaszolhat* (Response).

A Parancsok és Válaszok a legegyszerübb esetben az Információs mező kihagyásával, és a Vezérlés-jel megfelelő beállitásával jönnek létre. Az ellenőrzés a nyitó Zászló utáni teljes részt vizsgálja, - tehát a *vezérlő és a hasznos információ is azonos, hatékony hibakorlátozási eljárás* alá esik.

Az eljárás nagy előnye a teljes kódfüggetlenség, az *Információs mezőben tetszőleges kódolás alkalmazható.* Az eljárás lényegében csak a Keret-szerkezetet köti meg, ezért tetszőlegesen bővithető és variálható, a hagyományos eljárásoknak megfelelő "párbeszédes" forma is megvalósitható, de folyamatos, ellenőrzött átvitel is lehetséges az összetartozó Keret-ek sorszámozása utján.

---

[1] Mivel a szaknyelvben egyöntetű magyar kifejezés még nem honosodott meg, az eredeti angol nyelvű jelölést használtuk.

# 7.

# Terminálok

A terminálok teremtik meg a távadatfeldolgozó rendszer és a környezete közötti fizikai kapcsolatot, bonyolitják le közöttük az információcserét. Ennek megfelelően igen fontos, hogy mind az adathálózathoz, mind a környezethez jó hatásfokkal illeszkedjenek.

Ezek általában ellentétes követelmények, igy kielégitésük csak kompromisszumok árán lehetséges. /Irod. 2., 8., 9., 10., 11., 13., 14./

A távadatfeldolgozó rendszerek kialakulásának kezdetén részben a helyi feldolgozásban megszokott perifériális berendezéseket, részben a postai adatátvitelben alkalmazott végkészülékeket kapcsolták adatvonalakon át egy központi számitógéphez. Az első esetben a periféria vezérlő egységet járulékos vezérléssel kellett kiegésziteni és illeszteni kellett az adatvonalhoz. A második esetben ilyen jellegü feladatokat egy igen egyszerü logikai és illesztő egység már végzett, kisebb módositásokra azonban többnyire itt is szükség volt.

A kimondottan adatvégállomási feladatok ellátására tervezett *terminálok* felépitése, logikai vezérlése már olyan, hogy az időközben kidolgozott - ajánlott illetve szabványos - adatátviteli eljárások valamelyikével a terminál távvezérlése könnyen elvégezhető legyen.

A számitástechnika fejlődése során előtérbe kerültek az ember és a számitógép kapcsolatának megjavitására irányuló törekvések. A kapcsolat javitásának részfeladataként, egyre nagyobb volt az igény arra, hogy a gépet használó ember a számitógéppel ne valamilyen másodlagos adathordozó - lyukszalag, lyukkártya, mágnesszalag stb. - közbeiktatásával, hanem a számára legkedvezőbb formában megjelenő információ - irott szöveg, rajz, emberi hang - utján kommunikáljon.

Ugyancsak növekedett a szerepe a számitógéppel nagy távolságból történő közvetlen adatgyüjtésnek, méréspontellenőrzésnek, folyamatirányitásnak is.

Ezek az igények igen nagy lendületet adtak a terminálok fejlődésének.

Az egész rendszer megbizhatóságának fokozása céljából az utóbbi időkben tervezett terminálok többsége hibavédelmi berendezést is tartalmaz. Sok terminál képes arra, hogy bonyolult "intelligens" funkciókat, előfeldolgozási munkákat is ellásson.

Napjainkban már a terminálok igen széles választéka szerezhető be a világpiacon. Nem csupán a számitógép gyártó cégek, hanem rengeteg kis vállalat is foglalkozik a legismertebb számitógéptipusokhoz és legelterjedtebb adatátviteli vezérlési eljárásokhoz hatékonyan illeszkedő terminálok gyártásával és eladásával.

A megfelelő számitógép kiválasztása - ez szinte már közhely - igen komoly előtanulmányokat és ismereteket igényel. Egy távadatfeldolgozó rendszerben azonban - a központi számitógép kapacitásától is függően, esetleg már kis számu terminál müködtetése esetén is - *domináló lehet a terminálrendszer költségkihatása.*

A terminálok tulajdonságai ugyancsak visszahatnak a központi számitógép és az adatátviteli hálózat kihasználtságára, az egész rendszer hatékonyságára, és a környezethez, az emberhez való illeszkedésnek is legfontosabb tényezői.

Ezért általában a megfelelő terminálrendszer kiválasztására hasonlóan nagy sulyt kell helyezni, mint magára a központi számitógépre!

A terminál vonalcsatlakozón keresztül kapcsolódik az adatátviteli hálózathoz, együtt alkotják az adatvégállomást (ld. 1. ábra).

A vonalcsatlakozó felépitése elsősorban a csatlakozó adatvonal jellemzőitől függ.

A *terminál egység* hozza létre a rendszer-környezet illesztését, a tényleges rendszer input-outputot. A környezet lehet bármilyen - analóg vagy digitális - berendezés, amellyel a terminál egység fizikai jelek, másodlagos adathordozók utján cserél információt.

A mai terminál orientált on-line rendszerekben azonban a leggyakoribb "környezet" maga a számitógéppel - egyre gyakrabban interaktiv kapcsolatban - együttmüködő ember. Fizikailag a terminál egység lényegében megfelel a megszokott perifériális egységnek, bár számos uj, különleges - például teljesen elektronikus - tipus létezik.

A terminál *vezérlő egység,* - ahogyan elnevezése is mutatja, - ellátja a legtöbbször még részben mechanikus müködésü terminál egység vezérlését, a mechanika és elektronika "szinkronizálását".

Ezek a funkciók természetesen terminál tipusonként eltérőek, és tulajdonképpen nem különböznek a periféria vezérlő egységek hasonló feladataitól. A terminál vezérlő egységnek azonban az adatátvitel vezérléséhez kapcsolódó feladatai is vannak. Az adatátviteli vezérlési feladatokat a terminál vezérlő egység azonban rendszerint nem program, hanem operátori beavatkozások, illetve az üzeneteket megelőző vagy az üzenetek részét képező, az adatvonalon érkező adatátviteli vezérlő karakterek hatására fejti ki.

A terminál vezérlő egység a bitsorosan érkező információt fogadja, átmenetileg tárolja, összerakja, értelmezi.

Az üzenet részeit a vezérlő karakterek kódjai alapján szétválasztja, a felesleges részeket elhagyja, mint alállomás, részt vesz a párbeszédes adatátviteli eljárás lebonyolitásában, vezérlésében.

Az előző fejezetben az aszinkron és szinkron adatátvitel vezérlésének vizsgálatánál láttuk, hogy a terminál milyen jellegü válaszokat generál az adatátviteli vezérlő egységgel folytatott dialógus során. A hibavédelmi logika is igen gyakran a terminál vezérlő egység része.

A legegyszerübb terminál vezérlő egység csupán egyetlen terminál egységet illeszt és vezérel, igen gyakran célszerü azonban több terminál egységet együtt müködtetni. A közös vezérlő egység dolgozhat ugy, hogy egyidőben csak egyetlen terminál egységet szolgál ki és a többieket várakoztatja.

Feladatát azonban elláthatja bitmultiplex vagy karaktermultiplex módon is. Ez utóbbi két esetben minden terminál egység egyszerre müködhet, a felhasználók nem érzik, hogy várakoznak egymásra.

Ez lényegében azt jelenti, hogy a terminál egységek és a terminál vezérlő egység közötti adatáramlást a terminál vezérlő egység "távolvégi multiplexeléssel" egy nagyobb sávszélességü közös adatvonalra adja és igy juttatja el a központi feldolgozó számitógéphez.

Az is előfordul, hogy az input üzeneteket a terminál vezérlő egység átmeneti tárolás után a beérkezési sebességnél nagyobb adatátviteli sebességgel továbbitja a központi számitógép felé, adatkoncentrálást végez.

A terminál input-output egységek vezérlésének feladata megoldható egy - esetleg igen bonyolult - hardware vezérlő egységgel is, de napjainkban egyre jobban terjednek a ***programozható, "intelligens" terminál vezérlő egységek.***

A terminál vezérlő egység és a terminál egység közös szekrényben is elhelyezkedhet, egybeépülhet, ha azonban a közös vezérlő egység több terminál egységet szolgál ki, akkor általában külön szekrényben, saját tápegységről müködik.

A központi számitógép közelében is elhelyezhetünk terminálokat, például egyetlen épületen belül müködő time-sharing rendszerekben az ilyen igény gyakori. Ebben az esetben az adatátviteli vonalak olyan rövidek lehetnek, hogy a vonalcsatlakozót egyszerü kábel adó-vevő áramkörökkel is realizálhatjuk. Ilyen telephelyi rendszerekben gyakran alkalmaznak mind nagysebességü soros, mind bit-parallel átvitelt. A bit-parallel átvitelnél a terminál vezérlő egység lényegesen egyszerüsödhet. Az ilyen speciális esetekkel most nem foglalkozunk.

## *7.1. A terminálok csatlakoztatása*

A vonalcsatlakozó egység a terminált az adatátviteli vonalra illeszti.

A vonalcsatlakozó egység felépitését az adatvonal tulajdonságai, az adatátvitel módja határozza meg.

A csatlakoztatás paramétereit általában az adatátviteli hálózat tulajdonosa, a posta részletesen előirja, nemzetközi szabványokhoz igazodó ajánlásokban rögziti. Néhány országban bizonyos vonaltipusokhoz csak a postai szervektől bérelt vonalcsatlakozók használhatók, de ha nem igy van - mint például hazánkban - *a felhasználó vonalcsatlakozója akkor is minden esetben a postai előirásokhoz igazodik és postai engedélyhez kötött.*

A vonalcsatlakozó kiképzését befolyásolja, hogy az összeköttetés szimplex, félduplex vagy duplex áramkörön, kettő vagy hény huzallal jön-e létre.

Lényeges, hogy a terminált vonalkapcsolt vagy vonalkapcsolás nélküli összeköttetést biztositó hálózatra illesztjük-e.

Közvetlen összeköttetést adó *bérelt táviróvonalhoz a táviró vonalcsatlakozót használjuk.*

A *nyilvános telexhálózat* vonalcsatlakozója a *telex központhivó.* Ha ez egyszerü távgépirót illeszt a telexvonalhoz, akkor nem feltétlenül alkalmaznak hibavédelmet az adatvégállomáson.

Egyetlen központhivóhoz átkapcsoló segitségével több távgépiró is csatlakozhat.

Ha nem telex távgépirót, hanem más tipusu terminált illeszt a központhivó, akkor az illesztéshez *telex szimulátor* szükséges.

A *nyilvános távbeszélő hálózathoz* a legegyszerübb esetben egy - célszerüen nyomógombos - távbeszélő készülékkel csatlakozhatunk.

Ha a tiz számjegyhez rendelt nyomógombot még néhány - ez általában kettő - vezérlő nyomógombbal is kiegészitjük, akkor a telefonkészülékkel digitális információt billentyüzhetünk az adatvonalra. Az igy feltett kérdésre a készülék hallgatóján keresztül kaphatunk választ. Ez az egyszerü vonalcsatlakozó igy tulajdonképpen egy komplett terminál.

A telefonkészülékhez ugynevezett *akusztikus csatolásu terminált* kapcsolhatunk. Ebben az esetben a manuális kapcsolatteremtés után a kézibeszélőt akusztikus kapcsolatba hozzuk a terminállal. Ez a módszer főleg az Amerikai Egyesült Államokban terjedt el, a postai szervektől való függetlenités céljából.

A nyilvános távbeszélő hálózathoz általában *modemmel* csatlakozik az adatvégállomás. Ilyenkor is müködhet távbeszélő készülék a modem mellett. Az adat-, illetve távbeszélő üzemmód átkapcsolás a modembe van beépitve.

A modem *automatikus hivásfogadással* is kiegészithető.

Ebben az esetben a *hiváskezdeményezést manuálisan,* a telefonkészülék segitségével végezzük, a bejövő hivás alapján azonban az összeköttetés automatikusan felépül.

Sok esetben a hibavédelmi logikát - amit telefonvonalakhoz általában alkalmaznak - egybeépitik a modemmel.

A telefonvonalakhoz csatlakozó adatvégállomási modemet *automatikus hiváskezdeményezővel* viszonylag ritkán, főleg *intelligens* terminál vezérlő egység illesztésekor egészitik ki.

A szélessávu analóg adatvonalakhoz is modemen keresztül csatlakozunk, szélessávu áramkört azonban terminálok felé csak ritkán épitenek ki. Ilyennel például akkor találkozhatunk, ha a terminál vezérlő egység feladatait egy egyszerre több terminál egységet kiszolgáló kisszámitógép látja el.

A legujabb kapcsolt digitális adatátviteli hálózatokhoz valamilyen digitális vonalcsatlakozó illeszti a terminált. Ilyen egység például a *datex központhivó*, amely egyszerü hivott azonositást is végez.

Ujabban nem kapcsolt digitális hálózatokhoz alkalmas PCM vonalcsatlakozó egységek is megjelentek a világpiacon.

## *7.2. Terminálok jellemzői*

Már utaltunk arra, hogy a terminálok igen gyors fejlődésnek indultak az utóbbi évek során. Ez a fejlődés mind a terminál egység által lebonyolitott input-output tevékenység, mind a terminál vezérlő egység által végrehajtott funkciók, mind pedig a választék bővülésének tekintetében szembeszökő.

A "terminálok erdejében" meglehetősen nehéz eligazodni, nehéz egységes elvek alapján csoportokat képezni.

A berendezéseket igen sok szempont szerint lehetne fajtákba sorolni, sokféleségük miatt azonban aligha található olyan felosztás, amelynél ne lenne szükség az "egyéb" vagy "különleges" kategóriára.

A terminálok némelyike csak inputra vagy csak outputra alkalmas, mások mindkét irányban kommunikálhatnak. Eszerint beszélhetünk *input-*, *output-*, illetve *input-output* terminálokról.

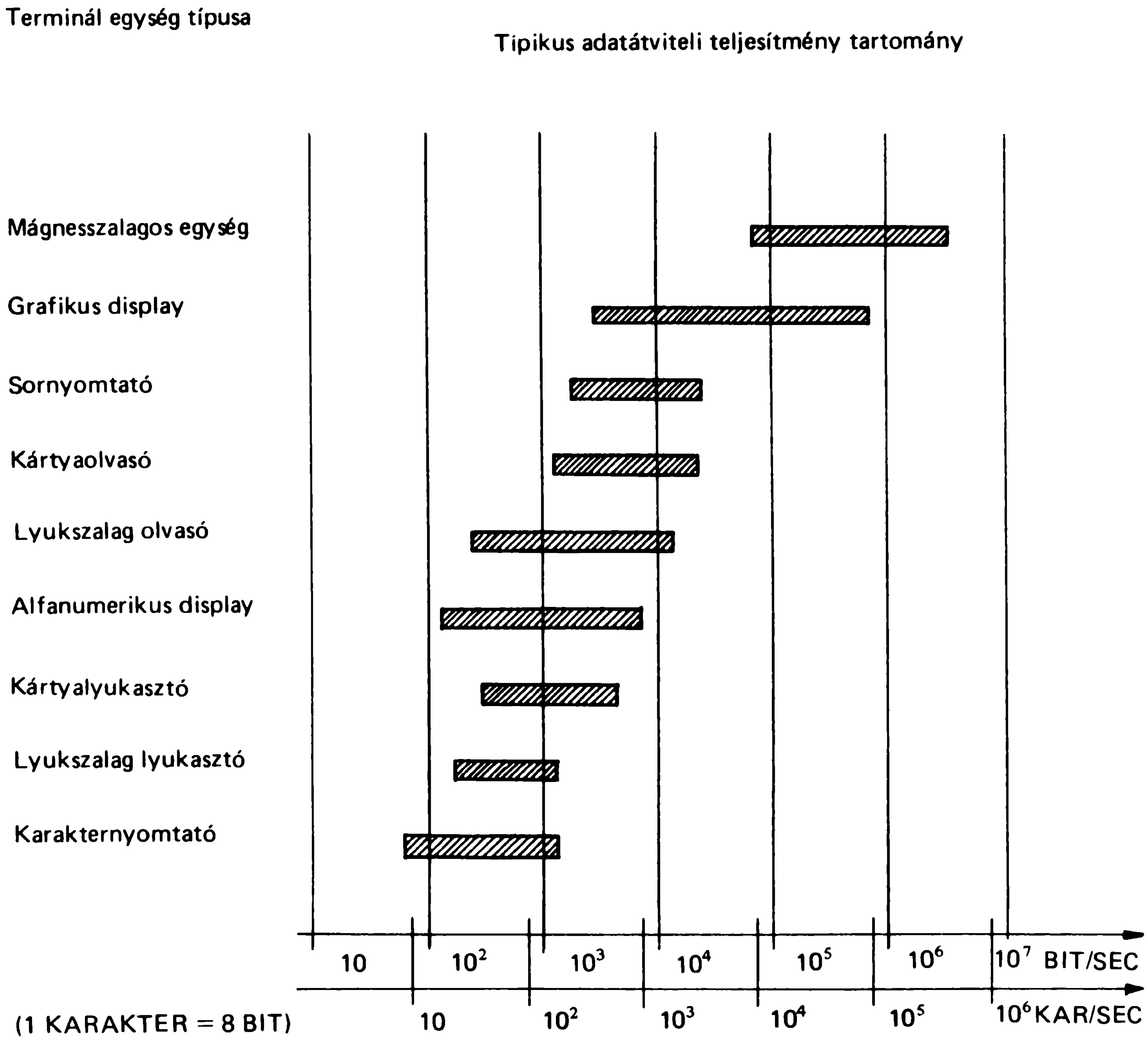
*Müködési sebességük* igen széles tartományt ölel fel. A terminál input-output egységek ma még többségükben elektromechanikus szerkezetek, ezeknél a mechanika a müködési sebességet korlátozza. Más esetekben a müködési sebesség korlátja a terminálnál ülő ember cselekvési vagy felfogási gyorsasága. Igen sok esetben limitálja a müködési sebességet az adatátviteli vonal sávszélessége. A terminálok müködési sebességét általában karakter/szekundumban adják meg és adatátviteli teljesitménynek nevezik. Ez néhány esetben (pl. grafikus terminálok, hangválasz berendezések) nem értelmezhető közvetlenül, hanem csak a bit/szekundum-os adatátviteli sebességen keresztül. A 16. ábra néhány gyakran használt terminál fajta adatátviteli teljesitményét hasonlitja össze.

Igen különbözőek lehetnek a terminálok az információ *megjelenitésének*, illetve *észlelésének módja*, az alkalmazott *adathordozó* szempontjából is.

A ki-, és bemenő információ lehet analóg vagy digitális jel, irott szöveg, rajzolt ábra, hangjel stb.

Adathordozónak a perifériáknál megszokott hagyományos fajtákat, illetve ezek továbbfejlesztett változatait használjuk:

lyukszalag, lyukkártya, mágneses felület, bizonylat, nyomtatványlap, vizuális kijelző felület stb.



*16. ábra*
*Terminálok adatátviteli teljesitménye*

A terminál lényeges tulajdonsága, hogy mennyi információt képes átmenetileg tárolni.

*Pufferelt terminálnak* csak azokat a berendezéseket tekintik, amelyek több karakternyi - például egy gépelt sornyi - információt képesek összegyüjteni és átmenetileg tárolni.

Különösen a közös adatvonalon felfüzött terminálok esetében fontos a pufferelés, ugyanis az adatvonal kihasználtságának hatékonysága igy lényegesen fokozható.

Ha a terminálnál ülő felhasználó "párbeszédes" ember-gép kapcsolatot épit ki a terminál segitségével, akkor *interaktiv terminálról* beszélünk.

Mint már emlitettük, ha a terminál vezérlő egység programozható, akkor *intelligens terminál* az elnevezése.

A programozható terminál vezérlő egység szerepét betöltheti akár egy erre a célra kifejlesztett, akár egy univerzális kisszámitógép. Az intelligens terminál jellemzője még, hogy az adatátvitel és a terminál input-output egység(ek) vezérlési feladatán tulmenő, járulékos feladatokat is végez (ld. 10. fejezet).

A fentiekkel már bizonyára éreztettük, hogy egészen egyszerü berendezésektől az igen bonyolultakig s igy a viszonylag olcsó terminálóktól a számitógépek árkategóriájába tartozókig terjed a választék.

Az alábbiakban megkiséreljük, hogy a különböző terminálokat viszonylag jól elkülönithető csoportokba soroljuk:

1) karakternyomtató terminálok,
2) alfanumerikus display terminálok,
3) grafikus display terminálok,
4) remote batch terminálok,
5) ügyviteli terminálok,
6) real-time terminálok,
7) különleges terminálok.

A fenti csoportok a perifériák hasonló feloszthatósága miatt első közelitésben önmagukért beszélnek.

Ahelyett, hogy - hely hiányában - az egyes terminálfajták specifikumaival foglalkoznánk, tekintsük át röviden a terminálok fejlődési irányait.

Mindenekelőtt, nagy intenzitással folynak a kisérletek az *ember és terminál jó hatásfoku illesztésére.* Ennek jegyében optimális elhelyezésü, kis erővel müködtethető billentyüzetekkel ellátott terminálok jelennek meg. Egyre több berendezés alkalmas arra, hogy *müködtetőjét azonositsa*, és automatikusan figyelmeztesse, ha hibát vét. Növekszik az *interaktiv pufferelt terminálok részaránya.* Ezek megkönnyitik, hogy a felhasználó üzeneteit csak véglegesre szerkesztett formában közölje a feldolgozó központtal, s ugyanakkor a közös adatvonalak jobb kihasználását is szolgálják.

Sok alkalmazásban előtérbe kerülnek a felhasználót *vizuálisan informáló alfanumerikus és grafikus display*-k, amelyekhez azonban opcionális másoló készülékek csatolhatók, hogy a felhasználó kívánságára papiron vagy mikrofilmen rögzitsék a "képet".

Ugyanakkor egyre több, az *alkalmazáshoz is jól illeszkedő terminállal* találkozhatunk. Például az ügyviteli és információs rendszerekben alkalmazott terminálokon a feldolgozásnál gyakran ismétlődő részfeladatokhoz *funkcionális nyomógomb* tartozik, amely meggyorsitja a munkát.

*A bonyolult és költséges mechanikát háttérbe szoritja az olcsó és megbizhatóbb elektronika.* Mind a géppel vezérelt nyomtatás és rajzolás, mind a beolvasás egyre inkább tisztán elektronikus uton történik. Ennek megfelelően terjed a mátrix-elektrosztatikus elven müködő printerek és printer-plotterek, valamint az optikai és a mágneses elven müködő bizonylatolvasók terminálként történő alkalmazása.

A *vonalcsatlakozó* egység általában a *terminálok integrált részévé válik.* Ugyancsak egyre több terminált szerelnek fel hatékony, integrált áramkörökből felépült hibavédelmi egységgel.

A terminál által nyujtott szolgáltatások növekednek, a terminálok intelligenciájának olcsó processor-okkal történő fokozása eredményeképpen.

A fenti törekvések ellenére érvényesül az *árak csökkenésének tendenciája.* Ez egyrészt a technológia fejlődésének, másrészt a tömeges gyártásnak köszönhető. A tömeggyárthatóság alapja az, hogy az összes terminálok igen nagy százaléka hasonló feladatok ellátására (az ügyvitelgépesitésben - bankok, pénzintézetek, áruházak stb. - és az információs rendszerekben - szállodai, utazási stb. helyfoglaló rendszerekben) kerül alkalmazásra.

# 8.

# A feldolgozó központok hardware architekturája

A feldolgozó központ lokális hardware és software *erőforrások* komplexuma. A központ alapvető problémája ezeknek, és a központból elérhető távoli *erőforrásoknak* a *hatékony kiosztása* az egyes felhasználók között. /Irod. 10., 11., 15., 16./

Az erőforrás kiosztás (resource allocation) a modern számitógépekben a hardware és software elemek olyan szoros szimbiozisát tételezi fel, hogy azok szinte alig különithetők el. Ez már a tervezés fázisában igaz, a hardware és software tervezése az utóbbi időkben párhuzamosan, kölcsönhatásban futó folyamat, amelyet mindkét szakterületen jártas szakemberek végeznek, akik még a tervezés utolsó, realizáló szakaszában is szorosan együttmüködnek. Csak igy érhető el, hogy végül is olyan számitógépet kapunk, amelynek erőforrásai hatékonyan és ugy rendelhetők az egyes felhasználókhoz, hogy azok az *erőforrás kiosztást nem érzékelik*. A tervezés alapmotivumai a hardware-software *modularitásra* és az alkalmazott müködtető *mechanizmusok egységére* való törekvés.

A rendszer alapvető hardware erőforrásai: az *input-output rendszer*, a *központi egység*(ek) és az *operativ tár*(ak). A hardware erőforrásoknak egy adott pillanatban érvényes kiosztását logikailag a rendszer-software *intézi*, *ütemezi*. Ha azonban a hardware ezt nem segitené, akkor sok felhasználó esetén a rendszer-softwarenek annyi dolga lenne, hogy a rendszer "belehalna" a saját adminisztrációjába: alig maradna ideje a felhasználók programjainak feldolgozására.

A távadatfeldolgozó rendszerek általában sok felhasználót szolgálnak ki egyszerre, ezért különösen fontos a hardware erőforrások hatékony kiosztása.

A rendszer erőforrásainak kiosztása a felhasználók és programjaik között a hagyományos számitógéprendszerekben is megtörtént, ezt a folyamatot azonban a *statikusság* jellemezte.

Az erőforrás-kiosztást lényegében a programozó végezte, jó előre megtervezte, és az egyszer specifikált kiosztást csak a programok ismételt leforditásával lehetett megváltoztatni.

Később ez ugy módosult, hogy bizonyos erőforrásokat a rendszer csak a *betöltéskor (loading) rendelt* a felhasználói programokhoz. Ez lehetővé tette, hogy külön leforditott programok közös fileokat, input-output erőforrásokat használjanak. Nem volt azonban lehetőség arra, hogy a már betöltött programok esetében egyszerüen változtassunk az erőforrások hozzárendelésén.

A távadatfeldolgozó rendszerek többségében a központi számitógép erőforrásainak használatára vonatkozó *igények dinamikusan változva*, előre megjósolhatatlan időben és mértékben lépnek fel. Az erőforrások azonban végesek és költségesek, ezért igen fontos, hogy a rendszer jól gazdálkodjék velük. Hatékony kiosztásukra a programozó vagy az operátor képtelen lenne. Ugyanakkor az erőforrásoknak a rendszer által történő statikus kiosztása sem felelne meg, hiszen nem tenné lehetővé a változó igényekhez való rugalmas alkalmazkodást. Emiatt sok igény kielégitetlen maradna, holott a rendszer még képes lenne - belső átszervezéssel - további szolgáltatásokra. A megoldás utja az, hogy az erőforrásoknak csak kis töredékét osztjuk ki statikusan, nagyobb részét *térben és időben párhuzamosan és dinamikusan ütemezve* rendeljük az igényekhez.

Az erőforrások ilyen módon történő kiosztásával elérhető, hogy a rendszer minden egyes felhasználójának az az illuziója, mintha saját, külön számitógépe lenne. Az egyes felhasználók az igy nyert *virtuális számitógépeket* egymástól függetlennek érzik, bár megoldható a virtuális rendszerelemek kommunikációja is. A virtuális számitógép virtuális input-output rendszerekből, központi egységekből és tárakból áll, a rendszer-software programjai és a file-ok szintén lehetnek virtuálisak, nagyrészük közös használatban van.

A virtuális számitógépek hardware-függetlenek, abban az értelemben, hogy a felhasználónak nem kell az egyes hardware elemek fizikai tulajdonságaival, a tényleges hardware konfigurációval törődni. A virtuális rendszerelemek logikailag többet nyujthatnak a felhasználók összességének, mint az a rendszertől egy adott pillanatban fizikailag elvárható. A felhasználók ugy programozzák saját virtuális számitógépüket, mintha az egy fizikai gép lenne. Éppen ezért ugy érzik, mintha specifikus problémájuk megoldására orientált hardware és programrendszer segitené munkájukat.

Nem szabad azonban elfelejteni, hogy az egyes virtuális számitógépeket ugyanaz a fizikai rendszer valósitja meg. Ezért gondoskodni kell arról, hogy elkerüljük a *független felhasználók és programok közötti nemkivánatos egymásrahatásokat.* Ugyanakkor biztositani kell, hogy a közös használatban lévő file-ok és programok az arra jogosultak számára elérhetők legyenek. Ez, más szavakkal, azt jelenti, hogy az erőforrásokat a kizárólagosságra, titkosságra, illetve közös használatra vonatkozó igények dinamikus figyelembevételével és összehangolásával kell kiosztani. Ennek biztositása részben szintén hardware feladat.

Igen lényeges szempont a feldolgozó központ *megbizhatósága,* hiszen mind a hardware, mind a software hibák egész felhasználói közösség munkáját tehetik bizonytalanná, vagy lehetetlenné. A hardware megbizhatóságára már a tervezés fázisában nagy figyelmet kell szentelni, hiszen a feldolgozó központ rendszerint nagyobb hardware komplexitásu, mint a helyi feldolgozást végző központok. A hardware megbizhatóság fokozásának fontos módszere a *kritikus rendszerelemek multiplikálása.*

Ahhoz, hogy e röviden összefoglalt igényeknek a feldolgozó központ megfeleljen, eleget kell tennie néhány hardware követelménynek.

A legfontosabb hardware követelmények:

1) *Autonom input-output rendszer*, amely az input-output berendezések adatforgalmát a központi egységtől függetlenül, annak müködésével párhuzamosan, aszinkron módon, az operativ tár önálló használatának utján képes lebonyolitani.

2) *Fejlett*, változtatható tulajdonságokkal rendelkező, többszintü *megszakitási rendszer*, amelynek segitségével a feldolgozó rendszer figyelme explicite hivható fel saját erőforrásai és külső tényezők állapotváltozására.

3) *Nagykapacitásu, dinamikusan kiosztható, gyors operativ tár*, amelynél a kiosztott tártartományok védelméről is rugalmasan gondoskodhatunk.

A feldolgozó központ logikai magja nem a központi egység, hanem az operativ tár, a központi egység és az input-output rendszer vetélkedve osztozkodik a tár használatán.

Ezeket a kérdéseket kissé részletesebben is megvizsgáljuk.

További fontos, de nem részletezett hardware tulajdonságok:

4) *Gyors müködésü központi egység*, amely elsősorban az időveszteség nélküli, gyors válasz követelményét hivatott kielégiteni. A központi egység sebességének növelése a technológián kivül szervezési, párhuzamos müveletvégzési lehetőségek kihasználásával történik.

A gyors központi egység teszi lehetővé az egyik programról a másikra történő gyors átkapcsolást, vezérlésátadást, ami a távadatfeldolgozó rendszerekben igen gyakori feladat. Néha fizikailag is több, független központi egység dolgozik együtt.

5) *Különleges utasitások* szükségesek. Ezek egy része ugynevezett *privilegizált utasitás*. A privilegizált utasitásokat csak a rendszer-software használhatja, a felhasználói programok nem. Ide tartoznak egyes input-output utasitások, a tárvédelemmel és tárkiosztással kapcsolatos utasitások, a megszakitási rendszerre és az órajelekre vonatkozó utasitások stb. Amikor a központi egység *privilegizált állapotban* van, akkor minden utasitás használható, ha azonban normál állapotban van, - amikor is a felhasználó programja fut - akkor csak a szükitett utasitáskészlet érhető el, a privilegizált utasitásokra történő hivatkozás programmegszakitást eredményez. A különleges utasitások egy másik csoportja az *adatátvitellel kapcsolatos* munkákat megkönnyitő *utasitások készlete*. Ezek az utasitások a bitkezelést, a karakterkezelést és a kódkonverziót vezérelhetik, illetve az üzenetek sorbaállitásának megszervezését segitik elő.

6) Előnyös a *mikroprogramozott vezérlés*. A mikroutasitások egy - rendszerint csak olvasható - gyors, kis kapacitásu vezérlő tárban helyezkednek el. Segitségükkel a rendszer tulajdonságai flexi-

bilisen változtathatók, az összeadómü és a vezérlő egység regiszterei közvetlenül kezelhetők. Ez elsősorban az operációs rendszer számára kedvező, - például emuláció megvalósitására - bár egyes rendszerek megengedik, hogy a felhasználó mikroutasitások segitségével módositsa az input-output müveleteket, a megszakitási feltételeket vagy különleges üzenet szerkesztési feladatokat oldjon meg, uj utasitásokat generáljon.

7) Szükség van *órajelek előállitására*. Ezt általában egy realtime óragenerátor és több időtartam beállitó óragenerátor végzi, amelyek egymással összhangban müködnek. Feladataik a programok által felhasznált idők mérése, előre programozott beavatkozási, munkakezdési idők jelzése, időfüggő események bekövetkezésének figyelése, az ütemezés vezérlése, a végtelen ciklusok és egyéb blokkolási hibák megkerülhetőségének biztositása.

8) A rendszer legyen *bővithető*, ez ugy értendő, hogy a meglévő konfiguráció átalakitás nélkül engedje meg további háttértárak, input-output perifériák és adatvonalak, terminálok csatlakoztatását, vagy egyes rendszerelemek multiplikálását.

9) A hagyományos helyi perifériák és a szekvenciális, valamint a mozgófejes, közvetlen hozzáférésü háttértárak mellett rendszerint szükség van gyorsabb közvetlen hozzáférésü háttértárakra, például fix-fejes mágneslemezekre, mágnesdobokra is. (5-20 msec átlagos elérési idő.)

Néha egy "lassu" (8-10 μsec) *ferritgyürüs tömegtár* illeszkedik a tárhierarchiába az operativ és háttértárak közé.

## *8.1. Az input-output rendszer*

Korábban láttuk, hogy a feldolgozó központ *adatátviteli vezérlő egységen* keresztül csatlakozik az adatátviteli hálózathoz. A hálózathoz történő illesztést a termináloknál felsoroltakkal megegyező, az adatvonalak tulajdonságaitól függő *vonalcsatlakozók* végzik.

Az adatátviteli vezérlő egység lényegében ugyanugy jelenik meg a központi számitógép számára, mint bármelyik periféria vezérlő egység. *Most az operativ tár irányában történő adatforgalmat* vizsgáljuk, ez magában foglalja mind az adatátviteli vezérlő egység, mind a többi input-output berendezések adatforgalmát.

A teljes input-output adatforgalom nemcsak az adatátviteli vonalak igényei miatt növekszik meg egy távadatfeldolgozó rendszerben. Ennél jelentősebb forgalomnövekedést okozhat az a tény, hogy a *sok felhasználói és rendszerprogramot* az operativ tár elégtelen mérete miatt *gyakran kell "cserélgetni" az operativ-, és a háttértárak között.*

Az input-output rendszernek olyannak kell lennie, hogy a megnövekedett adatforgalom mellett is jól kiosztható erőforrást jelentsen.

Hasonló, de kevésbé kiélezett problémák helyi multiprogramozásnál is felléphetnek.

Az adatforgalom középpontja az operativ tár, amelynek azonban nemcsak az input-output rendszer, hanem a központi egység adatforgalmi igényét is ki kell elégiteni. A tár fizikai felépitése következtében azonban jóval lassubb, mint a modern technológiák segitségével felépithető központi egység vagy input-output rendszer, ezért ez a sebességkonfliktus csak valamilyen ügyes szervezési elv utján hidalható át. Ilyen szempontból még a negyedik generációs, félvezetős (monolitikus) operativ tárak sebessége sem kielégitő.

A nehézségek másik, sulyosabb oldala az, hogy a központi egység és az input-output rendszer egymástól függetlenül kiván hozzáférni az operativ tárhoz, sőt az lenne az ideális, ha minden egyes input-output folyamat közvetlen és független módon használhatná a tárat.

Az input-output folyamatok függetlenitésének feladatát az *input-output csatornarendszer* látja el. Egy input-output folyamat *kezdeményezésének joga a központi egységnél marad, de lebonyolitása már az input-output csatorna autonom hatásköre.* A folyamat lebonyolitásához ismerni kell az adatforgalom végpontjait, vagyis az input-output berendezés "cimét" és a folyamathoz rendelt *tárbeli cimtartományt.*

Az egyes input-output berendezések egyrészt rendszerint nem kivánnak egyidőben kommunikálni az operativ tárral, másrészt adatátviteli sebességük többnyire lényegesen alacsonyabb, mint az operativ tár sebessége, ezért lehet vezérlési funkciókat összevontan, a multiplexelés elvén, input-output csatornák segitségével megoldani. A több input-output csatornából álló csatornarendszer aggregált áteresztő képessége azonban már nagyobb lehet, mint az operativ tár hozzáférési sebessége. A két alapvető csatornafajta: a *szelektor* és a *multiplexer csatorna.* Mindkettő több berendezés átvitelét vezérelheti. A szelektor csatorna egyszerre csak egy alcsatornát valósit meg, egyetlen input-output berendezés adatforgalmát képes lebonyolitani. Egy vagy több blokk átvitelének idejére hozzá van rendelve egy adott folyamathoz, és eközben a rábizott többi berendezés számára foglalt. Fenti tulajdonsága miatt gyors berendezések, főleg háttértárak nagysebességü adatforgalmának lebonyolitására előnyös, ahol az átviteli igény igen gyors blokkokban (burst) jelentkezik. (A szelektor csatorna tulajdonképpen teljes input-output folyamatokat multiplexel.)

A multiplexer csatorna ezzel szemben több, rendszerint 16-256 alcsatornát valósit meg.

Az egyes alcsatornákkal csak egy rövid információegység - egyetlen byte, szótöredék vagy szó - átvitelének idejéig foglalkozik, majd áttér a következő alcsatornára. Minden alcsatornához célszerüen egy lassu berendezés csatlakozik. A multiplexer csatornához kapcsolt *adatátviteli vezérlő egység annyi alcsatornát foglal le, ahány adatvonalat vezérel.*

Az input-output csatornák a központi egység kezdeményezésére, de a későbbiekben tőle függetlenül és müködésével aszinkron módon az operativ tárból vett *csatornaprogram* alapján vezérlik az adatforgalmat. A csatornaprogram *parancsok* formájában tartalmazza az input-output folyamat lebonyolitásához szükséges információkat. A szelektor csatorna átveszi a tárból ezeket a parancsokat és egy adatátviteli folyamat vezérlésének idejére saját regisztereiben tárolja. A multiplexer csatorna számára azonban "gazdaságosabb", ha a sok saját regiszter helyett az operativ tár meghatározott rekeszeiből kapja a vezérlő parancsokat. Ez viszont jelentősen megnöveli a tárhivatkozások számát. Mindkét fajta csatorna egyszerre csak egy rövid információegység - például byte - átvitelének idejére veszi igénybe az operativ tárat. Az egyes csatornák az operativ tár használata szempontjából prioritási sorba vannak állitva, a szelektor csatornák prioritása magasabb, mint a multiplexer csatornáé.

Az input-output csatornák program alapján bonyolult folyamatok vezérlését végzik, ezért joggal tekinthetők speciális kisszámitógépeknek, *input-output processzor*oknak.

Ez lényegében azt jelenti, hogy az operativ tár használatáért *több fizikai processzor vetélkedik;* rendszerint egy, de néha több központi processzor, és általában több input-output processzor. A vetélkedés következménye, hogy a processzorok még lassubbnak látják az operativ tárat, mint amilyen az valójában.

Ha az operativ tárat - legegyszerübb megoldásként - egyetlen tármodullal realizáljuk, akkor a processzorok vetélkedési konfliktusa igen élesen jelentkezhet, mert a tár fizikai felépitése nem teszi lehetővé, hogy egy modulhoz egy időpillanatban egynél több processzor férjen hozzá. Ez a módszer mégis használható, mivel az input-output folyamatok többsége viszonylag lassu, és az input-output rendszer - bár gyors müködésre képes - viszonylag ritkán kér kiszolgálást. Az input-output tevékenység véletlen és aszinkron jellege miatt, az információvesztés kiküszöbölése céljából, mégis az input-output rendszernek szoktak magasabb prioritást adni a tárhoz való hozzáférési hierarchiában. Ha most egy ilyen prioritásu rendszerben felduzzad és felgyorsul az input-output adatforgalom - és ez távadatfeldolgozásnál gyakori jelenség -, akkor ez hosszabb időre is kizárhatja a központi egységet a tár eléréséből, s igy a programok átfutása jelentős mértékben lelassulhat.

Az is előfordulhat, hogy további input-output folyamatok nem képesek elérni az operativ tárat, s igy kielégithetetlen input-output adatforgalmi igények lépnek fel.

Kézenfekvő megoldás e probléma kiküszöbölésére, hogy *több, független tármodult alkalmaznak.* Minden tármodul önálló cim-, és tartalom regiszterrel, vezérlő logikával rendelkezik (pl. XDS-940-es számitógép). Ha egy sinrendszer segitségével minden processzor minden tármodult elérhet, akkor a processzorok által látott tár mérete nem változik. Szerencsés esetben igy minden processzor másik tármodult használ egy adott pillanatban, de előfordulhat az is, hogy valamelyik modult valamelyik nagy adatforgalmu processzor hoszszabb időre lefoglalja, és igy várakoztatja a többi igénylőt. Ennek áthidalására az egymást követő tárcimü rekeszeket különböző, például körforgásosan egymást követő sorszámu, tármodulokban helyezhetjük el. Igy a többnyire gyors, szekvenciális tárhasználatot folytató input-output processzorok nem zárhatják ki egy blokk átvitelének idejére egymást és a központi egységet a tármodulok használatából. Természetesen bonyodalmakat jelent, hogy a központi egység tárhivatkozásai nem mindig szekvenciálisak, és hogy több processzor egyszerre is igényelheti ugyanazt a tármodult, de ezek a nehézségek csak tárciklus-időnyi blokkolást jelentenek.

Ez a *fésüzéses* (interleaving) eljárás tovább finomitható, ha még arra is ügyelünk, hogy egy program utasitásai és operandusai a tármodulok által alkotott két különböző blokkban helyezkedjen el.

A tár moduláris felépitése arra is lehetőséget ad, hogy a tár és a processzorok közötti sebességkülönbséget csökkentsük. Ez ugy érhető el, hogy az egyes processzorok időben átlapolva fordulnak a soronkövetkező modulokban elhelyezkedő soronkövetkező cimekhez. Ezt az átlapolásos tárelérési technikát (overlapping) elsősorban a központi egység(ek) sebességének jobb kihasználására szokták alkalmazni.

## *8.2. A megszakitási rendszer*

A központi számitógép hatékony vezérlésének és jólszervezett müködésének feltétele, hogy egyes egységei gyorsan ***informálhassák egymást*** belső vagy külső körülményektől függő ***állapotukról.*** Az erőforrások kiosztása csak ilyen friss információk birtokában ***ütemezhető dinamikusan.*** Az állapotvizsgálat elvileg ugy is elképzelhető, hogy a rendszer periódikusan megvizsgálja önmagát. Minél bonyolultabb azonban a rendszer architekturája, és több felhasználóinak száma, annál összetettebb és időigényesebb belső vizsgálatra lenne szükség, s ezért a rendszer egyre jobban "befelé fordulna". Ez ugy kerülhető el, ha a rendszer csak akkor folytatja le a vezérlési folyamatának megváltoztatásához szükséges állapotvizsgálatot, amikor az állapotváltozás már létrejött. A megszakitási (interrupt) rendszer éppen az ilyen állapotváltozásokra hivja fel a központi egység figyelmét. A megszakitások figyelését azért kell a központi egységben koncentrálni, mert az az erőforrások kiosztásának hardware felelőse, és a többi processzorok - bár átmenetileg önállóan is müködhetnek -, nem teljesen függetlenek a központi egységtől. A szokásosnál függetlenebb processzorok alkalmazásával megszakitási rendszer nélkül is hatékonyan megszervezhető az erőforrások kommunikációja, ez a módszer azonban nem terjedt el.

Egy megszakitási feltétel bekövetkezése esetén a központi egység a feltételhez rendelt megszakitási vonalon jelzést kap. Erre megváltoztatja állapotát, vezérlési folyamatát. Leállitja a futó programot, és átadja a vezérlést a megszakitást kiszolgáló rendszerprogramnak. A programleállitás egyszerü hardware müvelet, a vezérlésátadás egy fix rekeszre való hivatkozással történik.

A fenti folyamat kritikus pontjai a következők:

a) egyidőben több, vagy legalábbis egy elkezdett megszakitáskiszolgálás alatt ujabb - megszakitás léphet fel;

b) a megszakitott program folytatásához szükséges paraméterek kimentése és visszaállitása időigényes.

Az első probléma megoldásához röviden vizsgáljuk meg, hogy - keletkezésük helye szerint - milyen megszakitási jelek léphetnek fel. (Pl. IBM System/360):

1) *gépi megszakitások:* ide soroljuk a hardware-hibák - pl. feszültségkimaradás, processzorok hibái, paritáshiba - által okozott megszakitásokat; 2) *program hibákból adódó megszakitások:* pl. müveleti kód, vagy aritmetikai tulcsordulás következtében; 3) *program feltétel megszakitások:* akkor keletkeznek, ha maga a futó program kér megszakitást, mert szüksége van a rendszer-software segitségére; 4) *külső megszakitások:* keletkezhetnek egy másik központi egység, az operátor, valamelyik hardware óra stb. kezdeményezésére; 5) *input-output megszakitások:* amelyek input-output adatforgalommal vagy berendezéssel kapcsolatos folyamatok végrehajtását, megszakadását, hibás befejezését jelentik le.

Ha a sokféle független feltétel okozta, valamennyi megszakitást hatékonyan szeretnénk kiszolgálni, akkor meg kell engedni, hogy egy megkezdett kiszolgálás is megszakitható legyen. Ezt ugynevezett *többszintü megszakitási mechanizmussal* valósithatjuk meg, ami lehetővé teszi a megszakitások egymásba ágyazását. Nyilvánvaló, hogy az egyes megszakitási feltételek nem azonos fontosságuak a szükséges beavatkozás sürgőssége tekintetében. Az lenne a legkellemesebb, ha minden feltételnek saját prioritása lenne, és csak a magasabb prioritásu események szakithatnák meg az alacsonyabb prioritásuakat. Ezt a gyakorlatban ugy realizálják, hogy az egyes megszakitásokat prioritási csoportokba sorolják. Minden csoport további prioritási sorrendbe rendezhető, vagy várakozásra itélhető egyenrangu eseményekre tagolódik. Igy csak növekvő fontossági sorrendben bekövetkező megszakitási feltételek okoznak többszintü megszakitást és a szintekre tagozódás sem lesz tulságosan mély.

A prioritási csoportok legtöbbször - csökkenő prioritás szerint - éppen a megszakitások előző 1-5 sorrendjének felelnek meg. Az eddig elmondottakat általában multiprogramozott kötegelt feldolgozásnál is elvárják a számitógéptől.

Dinamikusan változó külső és belső tényezők azonban csak akkor vehetők hatékonyan figyelembe, ha a fenti elveket rugalmasan alkalmazzuk és nagy sebességgel realizáljuk. Változó körülmények között ugyanis a *megszakitások fontossága is változhat.* Ez a probléma ugy oldható meg, ha a *megszakitási mechanizmust* a mindenkori igényeknek megfelelően, *programmal változtathatjuk.* A változtathatóságot egyrészt a *megszakitási jelek figyelésének engedélyezhetősége és letilthatósága,* másrészt a jelek *prioritási besorolásának átcsoportosithatósága* jelenti (pl. XOS-940; PDP-10 számitógépek).

Eközben azonban a végrehajtási sebességre is ügyelni kell. A végrehajtás sebessége ugy fokozható, ha minél több feladatot bizunk a hardware-re. Ezt a célt is szolgálják az egyes *processzorok állapotát leiró* és a processzor életét figyelő *állapotszavak,* amelyek gyors, regiszterek közötti müveletek segitségével transzferálhatók, gyorsan lekérdezhetők. A velük kapcsolatos müveletek elvégzésében segitenek a megszakitási rendszerre vonatkozó privilegizált utasitások is. Egy mikroprogramozható megszakitási rendszer mind a rugalmas változtathatóság, mind pedig a gyorsaság kritériumainak eleget tesz.

## *8.3. Tárkiosztás, tárvédelem*

A távadatfeldolgozó rendszerekben, ahol általában elvárjuk, hogy bármikor egyszerre több feldolgozó program legyen futtatható, aktiv állapotban és ugyanakkor a belső adminisztrációt ellátó rendszerprogramok is felduzzadnak, talán az operativ tárral való gazdálkodás okozza a legtöbb nehézséget. A gazdaságosan megvalósitható, elegendően nagy kapacitásu és sebességü operativ tár hiánya miatt csak közvetlen hozzáférésü, gyors háttértárak, valamint bonyolult rendszertechnikai és programozástechnikai szervezési elvek segithetnek, ha el akarjuk érni, hogy a felhasználók összessége számára megjelenő "virtuális tár" nagyobb kapacitásu legyen, mint az operativ tár fizikai mérete.

Az operativ tár kiosztása azért bonyolultabb a többi erőforrások kiosztásánál, mert a számitógép tárcentrikus felépitésü, és a tártól - egy részének kivételével - elvárjuk, hogy bármelyik tartományát bármelyik programhoz, bármelyik processzorhoz hozzá lehessen rendelni.

Mivel elegendő, ha egy program csak közvetlenül futás előtt kerül be az operativ tárba, gyors háttértárak alkalmazásával is sokat segithetünk.

Az operativ tárban csak 1-2 feldolgozó program helyezkedik el, és ha egy további programra van szükség, akkor az operativ tár és a megfelelő háttértár közötti gyors *programcsere* (swapping) utján a részlegesen feldolgozott program a háttértárba, a kivánt program pedig az operativ tárba kerül. Ez a megoldás csak akkor engedhető meg, ha a programok "sokáig" maradhatnak az operativ tárban, mivel a vezérlésátadás "hosszu" időt vesz igénybe, ezalatt a központi egység tétlenkedik. A programoknak a tárban való hosszu idejü tartózkodására azonban a felhasználói igények gyakoriságának és a programok számának növekedésével egyre kisebb a remény.

A helyzet valamelyest javitható, ha ugynevezett ujrahivható, (reentrant) programokkal dolgozunk. Az ujrahivható program tulajdonsága, hogy végrehajtás közben nem változik és a végrehajtásához szükséges információkat kivülről kapja. Ha a több felhasználó által igénybe vett programok, például rendszerprogramok, ilyenek, akkor elegendő, ha egyetlen példányban találhatók meg az operativ tárban és nincs szükség minden felhasználás előtt ujbóli behivásukra. Másrészt viszont, ha egy ujrahivható program futása megszakad, akkor a részlegesen feldolgozott programot nem kell háttértárra menteni, mielőtt felülirná egy fontosabb uj program, hiszen futás közben az ujrahivható program nem változik, ezért ujbóli behivással a háttértáron lévő eredeti példány is folytatható. Igy az ujrahivható programok lényegesen csökkenthetik a programcserék mennyiségét.

Ha azonban egyáltalán programcseréket kell végezni, felmerül a kérdés, hogy egy adott program az operativ tár *különböző tartományaiban* milyen feltételek mellett *használható ismételten.*

*A programok különböző tartományokba való áthelyezhetőségének lehetősége* nyilván szükséges, ha azt kivánjuk, hogy a tár dinamikusan, az éppen felmerülő igények szerint legyen kiosztható.

A programozóktól nem várható el, hogy abszolut tárcimekben gondolkozzanak, az általuk használt látszólagos tárat logikai cimtartományon át érik el. A logikai cimek fizikai cimekké való átalakitása (mapping) a hagyományos feldolgozásoknál a programok betöltése során (loading) történik. Ennek előnye, hogy a különböző programok futtatás előtt automatikusan összekapcsolhatók (linking). Ha ebben az esetben kellene egy részben feldolgozott programot egy másik tártartományba áthelyezni, akkor ez a már kialakitott abszolut cimek miatt csak a betöltési procedura megismétlésével volna lehetséges, ami természetesen nagy időveszteséget okozna. Ráadásul további nehézségek is fellépnek, ezért ez az ut nem kedvező. Mivel a logikai cimek átalakitása, a betöltés alatti elhelyezés végleges, a programok futás közben nem helyezhetők át, ezt a módszert *statikus áthelyezésnek* (static relocation) nevezzük.

Megoldható azonban, hogy a végleges logikai-fizikai cim átszámitás csak az utolsó pillanatban, a program végrehajtásakor történjen meg. Igy mód nyilik a *programok futás közben történő áthelyezésére*, ezért ezt az eljárást *dinamikus áthelyezésnek* (dinamic relocation) nevezzük. A dinamikus áthelyezés lehetővé teszi a tár dinamikus, ismételt, gyors ujrakiosztását. Megvalósitása azonban csak ötletes hardware architekturák intenziv hasznositásával oldható meg.

Ezeket a technikákat most hely szükében nem részletezhetjük, csupán néhány lényeges tulajdonságukat kiséreljük meg kiemelni:

1) *Bázisregiszterek alkalmazásával* egy programot dinamikusan az operativ tár bármely részében elhelyezhetünk ugy, hogy egy bá-

zisregiszterben megadjuk a tartomány kezdőcimét. A fizikai cimet a bázisregiszter tartalmának és a program relativ cimének összege adja. Tulajdonképpen a programok betöltése logikai cimekre történik, és a programhoz rendelt bázisregiszter mutatja meg a logikai-fizikai cimek kapcsolatát. A programot teljes egészében be kell vinni a tárba, ezért a programiráskor felhasznált logikai cimtartományt a fizikai cimtartomány korlátozza. Egy háttértárra vitt, részlegesen feldolgozott program folytatásához az szükséges, hogy áthelyezéséhez elegendően nagy *összefüggő tartományt* találjunk az operativ tárban. Ha ilyen nincs, akkor a tárban lévő többi programot át kell rendezni, vagy valamelyiket részben vagy egészében háttértárra kell vinni. Ha egy program különböző részeihez különböző bázisregisztereket rendelünk, akkor a program nem összefüggő tártartományokban is elhelyezhető.

Speciális átfedéses programszerkesztéssel (overlaying) a felhasználó elérheti, hogy logikai cimtartományát nagyobbnak érzékeli, mint a rendelkezésére álló fizikai cimtartomány.

2) *Laptechnika* alkalmazásával kiküszöbölhető a bázisregiszteres technikának az a hátránya, hogy viszonylag *nagy, üres tártartományok keletkezhetnek*, amelyek azonban nem elegendően nagyok egy ujabb teljes program befogadására. Ezt a nehézséget az összefüggő nagy tartományok áthelyezése nem oldja meg kielégitően.

Laptechnikánál a tárat kisebb, azonos méretü (0, 5 - 4 KByte) *blokkokra*, a programokat pedig ezekkel megegyező méretü *lapokra* hasitjuk. *A logikailag sorban egymás után következő lapokat fizikailag* az üresedéseknek megfelelően *szétszórt blokkokban* helyezzük el. Az elhelyezés a rendszer feladata, a felhasználó nem érzékeli.

Nem szükséges, hogy a futó program minden lapja egyszerre a tárban tartózkodjék, ezért a programozó által használt *logikai cimtartomány jóval meghaladhatja* a fizikai tárméretet.

A gyakorlati megvalósitáshoz minden programnak rendelkeznie kell egy *laptáblával*, amely a cimátszámitáshoz szükséges, *dinamikusan változtatható utmutatásokat* tartalmazza. A laptábla hardware igényes; elhelyezkedhet magában az operativ tárban is, de hatékonyabb, ha külön *gyors regisztereket*, vagy kiskapacitásu *asszociativ tárat* biztositunk számára.

Laptechnikával az operativ tár jobban kihasználható, mint bázisregiszterekkel. A lapok cseréjének ideje alatt a központi egység általában talál magának hasznos elfoglaltságot, néhány esetben azonban ezek a potenciális előnyök nem aknázhatók ki olyan mértékben, hogy ez egyensulyban lenne a szervezés bonyolultságával, időigényességével és árával.

3) A *szegmentálás* bevezetését az tette szükségessé, hogy bizonyos *program és adattartományok* megkövetelik, hogy *feldolgozás közben szabadon és dinamikusan változtathassák méreteiket*. Az előző módszerek alkalmazásakor ilyen esetekben nagy tartalék tárterületeket kell rezerválni, amelyek azonban kihasználatlanok marad-

hatnak. Fellép az az igény is, hogy programokat és adatokat *közös használatba* bocsássunk. A felhasználók azonban ezeket egymástól különböző és változó módon kivánják használni, ezért ugyanahhoz a fizikai területhez többféle logikai elérési lehetőséget kell dinamikusan biztositani. A szegmentálás ezeknek a nehézségeknek a leküzdésében segit.

A bázisregiszteres- és a laptechnika sem tisztán hardware módszer, a szegmentálás azonban mindenekelőtt szervezési eljárás, amely lényegében az előző technikákra jellemző hardware elemekkel megvalósitható.

A *szegmens* egy olyan logikai cimtartomány, amelyben egy programrész vagy adathalmaz helyezkedik el. A szegmensekből programok, file-ok állithatók össze. A szegmentálás során nem a fizikai tárat, hanem egy igen nagy *logikai cimtartományt osztunk ki.* Ehhez egy kétdimenziós cimzési eljárást használunk. Relativ cimzést végzünk a logikai cimtartományban. A logikai tartományokhoz azután fizikai tárkiosztás, például laptechnika segitségével fizikai tartományok rendelhetők.

A kiosztott logikai cimtartomány a tényleges fizikai, operativ tár-beli cimtartomány sokszorosa lehet.

A szegmenseket *szegmensnévvel* látjuk el, minden szegmenshez egy *szegmens leiró szó* tartozik. Egy program az általa használt szegmensek leirását a hozzárendelt *szegmens leiró táblázatban* találja meg. A leiró információk dinamikusan változtathatók, és a *végleges összerendeléseket elegendő futáskor elvégezni.* Különböző logikai cimtartományokhoz azonos fizikai cimtartományok tartozhatnak más elérési paraméterekkel. Nagyméretü üres logikai tartományok sem okozzák üres fizikai tartományok keletkezését.

A szegmentálás "megszervezéséhez", a leiró információk és az ezek előkereséséhez szükséges katalógusok tárolásához és hatékony használatához gyors kisegitő hardware használata előnyös. Ehhez társul még a részproblémaként jelentkező fizikai tárkiosztás realizálásához szükséges speciális hardware architektura.

A fenti módszerek bármelyikének alkalmazásakor a fizikai tárat külön-külön használt blokkokra bontjuk.

Az éppen aktuális tárkiosztásnak megfelelően biztositani kell, hogy ezeket a blokkokat csak az arra illetékes felhasználói-, vagy rendszerprogramok érhessék el és csakis előre meghatározott módon. A rendszerben lehetnek olyan tárterületek, amelyekhez a felhasználók egyáltalán nem férhetnek hozzá (inaccessible), lehetnek olyanok, amelyek olvashatók és felülirhatók (read-write), amelyek csak olvashatók (read only) és olyanok, amelyek tartalma csak programként müködtethető, (execute only). Ha egy-egy blokkhoz egy hardware regiszterben az *elérés módját jelző információt rendelünk,* akkor a blokk vagy "mindenki" számára azonos módon lesz elérhető, vagy minden elérés előtt időigényes software beavatkozásra lesz szükség, amelynek során az elérési mód megváltoztatható.

Ennek továbbfejlesztése az a módszer, amikor az elérési módot leiró bitekhez *záróbitek* is társulnak.

Az elérési és záróbitek is dinamikusan változtathatók. Minden elérési kisérlethez ismerni kell a zár nyitásához szükséges "kulcsinformációt", ennek hiányában a kisérlet programmegszakitást eredményez. Az ilyen hardware zárak variációinak száma azonban alacsony, ezért a fenti védelmi eljárást még software "trükkökkel" szokták kiegésziteni.

# 9.

# A központi számítógép néhány software kérdése

A távadatfeldolgozással kapcsolatos software problémák nagyrészét a konkrét alkalmazás speciális kérdései jelentik. Az on-line távadatfeldolgozó rendszerekben a sok felhasználó és gyakori válaszigény következtében fellépő, valamint a minden esetben jelentkező kommunikációs software kérdések megoldása is a feldolgozási módtól és az alkalmazástól függ, de azért ezek közös jellemzői kiemelhetők. Még a dolgok ilyen leegyszerüsitése sem teszi azonban lehetővé, hogy e szükre szabott fejezetben tullépjük az alapvető software problémáknak a teljesség igénye nélküli felvetését. /Irod. 15., 16./

Mint az előző fejezetből is érezhető, a *rendszer-software* - részben mint *erőforrás*, részben mint a hardware-software *erőforrások kiosztásának mechanizmusa, intézője*, - a felhasználót szolgálja. A felhasználók szolgáltatásigénye általában különböző. Egymással párhuzamosan is jelentkezhet, és sok esetben *gyakoribb*, mint a *kiszolgálások átlagos ideje*. Ennek következtében *várakozási sorok*, *befejezetlen kiszolgálások*, *részlegesen feldolgozott programok* keletkezhetnek, a rendszernek azonban biztositania kell, hogy mindez a felhasználót ne zavarja.

Ez csak ugy lehetséges, ha a rendszer-software olyan gyors és dinamikus hardware-software összjátékot teremt, amelynek során a *virtuális hardware-software halmazokat, processzorokat logikai folyamatokhoz, processzékhez rendeli.*

A processzor ebben az értelemben nem feltétlenül esik egybe valamely fizikai, hardware processzorral, hanem bármely olyan, ennél bővebb hardware-software kombinációból álló logikai egység is lehet, amely utasitásokat, parancsokat képes végrehajtani.

A processz olyan egy, vagy több programból összeálló logikai egység, amely vezérli a vele egyértelmüen összerendelt processzort és tartalmazza annak definicióját is.

A felhasználók job-jai olyan önálló és függetlenül feldolgozható egységek, amelyek rendszerint több processz jól kézbentartható, hierarchikus, fa-strukturáju összerendeléséből állnak. A job-ok processz-szerkezete lehetővé teszi, hogy a rendszerben a programok szervezését és feldolgozását egyszerüen és a hardware korlátok legyőzésével, a függetlenül működtethető processzorok nyujtotta párhuzamos végrehajthatóság kihasználásával végezzük el. A rendszerben egy adott pillanatban annyi processz futhat, ahány fizikai processzorból a rendszer felépül. Ezek a processzek azonos és különböző felhasználói job-okhoz is tartozhatnak.

A rendszer-software főfeladata, hogy a processzorokat, valamint a processzeket összerendelje és dinamikusan a felhasználók igényeinek hatékony és védett kiszolgálását megcélozva, kiossza az időben. E kiosztás közben azonban nem csak az igényekre, hanem az eltelt időre is tekintettel lehet.

A felhasználók egy "software ablakon" át, *parancsnyelv* (command language) segitségével tartják a kapcsolatot a rendszerrel. Ezen keresztül érik el a rendszer erőforrásait, hardware és software szolgáltatásait, az operativ tárat a fizikai processzorokat, a file-rendszert, a feldolgozó rutinokat stb.

A felhasználókat azonban nem érdekli, hogy a rendszer hogyan osztja ki erőforrásait, "csupán" parancsai mielőbbi teljesitésére tart igényt.

A rendszer-software legjobban talán fontosabb részfeladatai tükrében mutatható be. E részfeladatok egy része olyan, hogy az általa reprezentált szolgáltatás használatára vonatkozó igények nagyon türelmetlenek vagy gyakoriak. Az ilyen részfeladatokat megoldó rutinokat állandóan készenlétben kell tartani az operativ tárban, ez a rendszer-software rezidens része (resident routines). A rezidens rész ütemezi a processzeket, osztja ki az operativ tárat és az input-output berendezéseket, kezeli a megszakitásokat és az input-output müveletekre - beleértve az adatátvitelt is - irányuló igényeket stb. A kevésbé sürgős és ritkábban igényelt szolgáltatásokat teljesitő rutinokat a felhasználó programjaihoz hasonlóan az operativ tár és a háttértárak közötti rendszeres programcsere és dinamikus áthelyezés jellemzik.

A cserélgethető rutinok által nyujtott szolgáltatások közé tartozik például a file-rendszer kezelés, a felhasználók naplózása és a számlázás, a feldolgozó programcsomagok kiosztása, a programforditás és a különböző rendszerkiszolgálások stb.

Ezek a rutinok célszerüen ujrahivható programok.

Tulajdonképpen az, hogy a rendszer-software melyik része rezidens és melyik nem az, egy alapvető rezidens vezérlőprogramkészleten tul, alkalmazás- és rendszer-függő.

A következőkben csak a számunkra legfontosabb rendszer-software feladatok néhány kérdését vizsgáljuk meg. Nem foglalkozunk a programozási rendszer, az interaktiv programnyelvek és interpretativ forditó programok problémakörével, amely kérdések inkább a feldolgozási módtól és az alkalmazástól függőek.

## *9.1. Az ütemező feladatai*

A rendszer-software egyik legfontosabb tevékenysége a feldolgozó programok futásának megszervezése, vagyis a rendszerben található *processzek ütemezése*. Ezt a feladatot látja el az *ütemező* (scheduler). Az ütemezés két okból igen bonyolult feladat:

a) a processzek száma jóval nagyobb, mint az egyidőben rendelkezésre álló fizikai processzorok száma;

b) a processzek tulajdonságai és igényei igen különbözők.

Az első tulajdonságból következik, hogy a fizikai processzorok használatát egyáltalán ki kell osztani az időben. Ez az oka annak is, hogy a rendszerben található összes processzek három különböző *állapotot* vehetnek fel:

1) futó processz,
2) futásra kész processz,
3) blokkolt processz.

A rendszerben eltöltött idő során a processzek állapota dinamikusan változik.

A második jellemző és a fenti állapotok léte kivánja meg, hogy az *ütemezés* a processzek tulajdonságainak, igényeinek és pillanatnyi állapotának *megfelelő dinamizmussal* történjen.

A futásra kész és a blokkolt processzek programjaikkal, adataikkal és a hozzájuk tartozó virtuális processzorok leirásával együtt célszerüen háttértárakon helyezkednek el. A központi tárban csupán a processzek leirásának minimális részét, mindenekelőtt az ütemezéshez szükséges információkat kell tárolni *állapot szavak* formájában Az ütemező ezeket az információkat *nyilvántartja*, és ezek, valamint a rendszerben létrejött változások alapján végzi el a processzek besorolását.

Az ütemező a processzek állapotának nyilvántartásán kivül a következő, jól elkülönithető feladatokat látja el:

1) A processzek állapotának és az ütemezési stratégiának megfelelően a futásra kész processzek között valamilyen *futási sorrendet állapit meg*;

2) *kijelöli a következő futó processzt*, miután valamelyik processzor felszabadul.

A rendszer állapotváltozásait *megszakitások* jelzik. A megszakitások *rendszerint közvetlenül hivják az ütemezőt*, hogy az a futási sorrend ujraértékelését minél előbb elvégezhesse. Ennek hátránya a tulságosan felduzzadó adminisztrációs holtidő (system overhead), ami ugy kerülhető el, hogy az ütemező rendesen csak előre meghatározott időnként értékeli ujra korábbi munkáját, és csupán a legfontosabb állapotváltozásokra reagál azonnal.

A futásra kész processzek esetleg egy rendezetlen, de inkább egy célszerüen rendezett *készenléti listán* (ready list) helyezkednek el.

A készenléti lista *egy várakozási sor*, melynek rendezése általában a processzekhez rendelt *prioritás alapján* történik.

Ez a módszer egyben meghatározza a listáról történő válogatás, futásra kijelölés, tehát az ütemezés stratégiáját is, amely mindenekelőtt a rendszerrel szemben támasztott *felhasználói követelményektől* függ.

A legegyszerübb esetben a processzek futtatása a készenléti listára érkezés sorrendjében történik, körforgásosan (round robin scheduling). Ennél bonyolultabb, ha minden processzhez fix prioritás tartozik.

Még bonyolultabb, ha a processzek *különböző prioritási szintü várakozási sorokat* (multilevel queues) alkotnak, amelyeket pointeres listaszerkezettel realizálunk. A processzek prioritása tulajdonságaik és az eltelt idő függvényében dinamikusan változik, tehát az egyes processzek is dinamikusan vándorolnak a várakozási sorok között. Ezzel a módszerrel biztositható a rövid, input-output igényes interaktiv processzek és a hosszabb számitásigényes processzek hatékony szimultán kiszolgálása.

## *9.2. Tárkiosztás software oldalról*

A tárkiosztás az a rendszer-részfeladat, amelynél a hardware és a software egymást kölcsönösen meghatározó jellege legerősebben kidomborodik. Márpedig a központi tár lehetséges hardware architektúráit is csak felületesen tekintettük át. Ennek ellenére néhány mondatban megkiséreljük összefoglalni a cim által takart alapgondolatokat.

Már az előző fejezetben kiderült, hogy a távadatfeldolgozó rendszerek többségében a tárkiosztás alapvető problémája a lehető legkésőbbi, a futás idejére halasztott dinamikus programáthelyezés.

Az operativ tárban helyet kell biztositani a rendszer-software rezidens részének, nem rezidens vezérlő és feldolgozó programjainak, a rendszer által kezelt várakozási soroknak, táblázatoknak, puffereknek és a felhasználók feldolgozó programjainak.

A rendszer-software rezidens részét fix tártartomány hozzárendelése jellemzi; egyes nem rezidens, de tranziens vezérlőprogramokat, táblázatokat stb. a rendszer statikus áthelyezéssel is használhat, általában azonban a tár nagyobbik részét a dinamikusan áthelyezhető rendszer-, és felhasználói programok között kell kiosztani.

Az operativ tártartományt igénylő logikai egység - az eddigiekkel összhangban - a processz. A feladat tehát - az operativ tár architekturájától függetlenül - a processzek és a megfelelő fizikai tárblokkok összerendelése.

A *tárkiosztó rendszerprogram* e feladat ellátása érdekében a következőket végzi:

1) nyilvántartja: a) az egyes processzek tényleges elhelyezkedését és
b) a szabad tárterületeket
a logikai és fizikai tárhierarchiában;

2) valamilyen *tárkiosztási stratégia* szerint eldönti, hogy a futásra váró processz, vagy annak egy része számára, hol és hogyan biztositson helyet, és előkésziti az áthelyezést.

A nyilvántartás során az egyes processzekhez tartozó virtuális tár és fizikai tár közötti kapcsolatot regiszterek, táblázatok (ld. laptábla) irják le.

A táblázatok "kitöltése" a tárkiosztó rendszerprogram feladata.

A tárak gyakorlati megvalósitási módszerei - mint ezt láttuk - különböznek. Ennek megfelelően a software feladatok is sokfélék.

A közös elvárások röviden a következők:

A nyilvántartás használjon kevés táblázatot, hardware-rel támogatott egyszerü kitöltési és gyors kiértékelési módszert.

A táblázatok ne foglaljanak el nagy tárterületet és biztositsák a processzek egymással szembeni megfelelő védelmét.

A *tárkiosztási stratégia*, aszerint, hogy egy futásra váró processz milyen nagy részének ad helyet a fizikai tárban, kétféle lehet. Ha a futás kezdetén a processz teljes logikai cimtartománya fizikai tárhoz jut, akkor a processz futása közben nem történhet uj tárterület kiosztását igénylő tárhivatkozás, de a virtuális tár kisebb lesz, mint a fizikai.

Ha több processzt kivánunk egyszerre az operativ tárban tartani, akkor általában szükséges - és mindenképpen előnyös -, ha nem kötünk le előre tul sok, átmenetileg felesleges tárterületet, hanem csak a processzek egy-egy részét (pl. lapját) visszük be a tár - változó vagy fix méretü - blokkjaiba. Az ilyen stratégiai alapelv alkalmazásakor a virtuális tár nagyobb kapacitásu, mint a fizikai tár. Természetesen többször előfordulhat, hogy egy processz futása azért szakad meg, mert a folytatás nincs a tárban. Akár fix, akár változó hosszuságu fizikai blokkokra épitett tárkiosztást alkalmazunk, előbb-utóbb a tár tele lesz, és az ilyen folytatások, valamint a sürgős processzek beviteléhez helyet kell felszabaditani. A "helycsinálás" a tárban lévő valamelyik blokk tartalmának kivitelével, a blokkok átrendezésével, programcserék lebonyolitásával történik. Igen fontos annak meghatározása, hogy melyik blokk tartalmát vigyük háttértárra. A legegyszerübb esetben a *legrégebben behivott* blokk tartalmát viszi ki a rendszer egy lista alapján (first-in first-out strategy). Ennél ügyesebb módszer az, amelyik a *legrégebben futott* processz által lefoglalt tárterületet teszi szabaddá (least recently used strategy).

A processzek között a tárért folyó káros versengést igyekszik elkerülni az a stratégia, amelyik legelőször a futó processz egy meghatározott "készleten" felüli részének kivitelével próbálkozik (working-set strategy).

Olyan eljárás is létezik, amely a processzek "előéletéről" készitett statisztika alapján határozza meg, hogy melyik processzel bánjon mostohán és melyiket részesitse előnyben a tárkiosztás során.

## 9.3. *A file-kezelő rendszer néhány jellemzője*

A file összetartozó információk névvel ellátott gyüjteménye. Az on-line távadatfeldolgozó rendszerek többségében szükséges nagy mennyiségü információt file-ok formájában - az operativ tár kapacitásának elégtelensége miatt - háttértárakon tároljuk.

A *file-kezelő rendszer* a software-nek az a része, amely a felhasználó számára oly fontos file-szervezéshez gyakorlati segitséget nyujt, kényelmes file-kezelést tesz lehetővé. Automatizálja a file-ok nyitását, generálását, változtatását, karbantartását, zárását és törlését, mely müveletekhez "kivülről" csak a felhasználó *makroutasitásokba* irt *paramétereit* igényeli.

A file-ok és az operativ tár "háztartása" tulajdonképpen egymással összefonódó folyamat. Ugyanakkor a file-ok kezelése egy általános input-output rendszer részeként is felfogható.

A hagyományos helyi adagolt feldolgozást végző rendszerekben sem kivánatos, de eltürhető, hogy a gépidő nagy részét a felhasználó és a rendszer a file-ok kezelésére forditsa.

Távadatfeldolgozási környezetben ez általában megengedhetetlen, mivel a file-ok irányából is biztositani kell az operativ tár és a háttértár területek jól szervezett, gyors cseréjét. Ez nem csupán a gyorsabb, közvetlen hozzáférésü háttértárak alkalmazásán, hanem a file-kezelő rendszer megfelelő kialakitásán is mulik.

A központi számitógép *file-kezelő rendszerével* szemben támasztott általános *követelmények* a következők:

1) Biztositsa a file-okhoz való *közvetlen hozzáférést*. A közvetlen hozzáférés a file-ok cimzési, leképzési rendszerén mulik, de természetesen csak közvetlen hozzáférésü háttértárak alkalmazásával valósitható meg hatékonyan. A közvetlen hozzáférésü file-ok lehetővé teszik - a hagyományos feldolgozásokban legelterjedtebb és a távadatfeldolgozásban rendszerint csak kiegészitésként alkalmazott szekvenciális hozzáféréssel szemben - az egyes logikai rekordok tetszőleges sorrendben történő elérését. Ez azon alapszik, hogy minden rekord logikai azonositója, *"kulcsa"* és a neki megfelelő fizikai rekord(ok) cime(i) között - ebben az irányban - egyértelmü kapcsolat van. A kapcsolatot valamilyen *kulcsleképzés* teremti meg. A sokféle létező leképzési eljárás közül azokat részesitjük előnyben, amelyek több szempont szerinti keresést és gyors hozzáférést tesznek lehetővé.

2) Tegye lehetővé a file-ok *megfelelő nyilvántartását*. A nyilvántartás *file-katalógusban* történik. Távadatfeldolgozó rendszerekben ez igen gyakran hierarchikus katalógusrendszerré fejlődik. Ez rendkivül rugalmas, szimbólikus file-név használatot tesz lehetővé. Mivel a felhasználó csak a file-ok logikai strukturáját látja, a file-ok automatikus, belső, fizikai átszervezéséről nem is szerez tudomást, számára a file-kezelés hardware-független. A katalógusrendszerben való keresés legyen gyors és egyszerü, és a felhasznált tár legyen minél kisebb.

3) A file-kezelő rendszer engedje meg, hogy az arra jogosult különböző *felhasználók közös file-okra hivatkozzanak*, épithessenek egymás munkájára.

Általában az is meghatározható, hogy melyik felhasználó milyen módon érheti el az egyes file-okat. A felhasználók jogainak kiosztása, korlátozása, a file-ok védelmi és titkositási mértékének meghatározása a file-tulajdonos hatáskörébe tartozik. Az ilyen jellegü paraméterek dinamikusan változhatnak és a file-hoz tartozó katalógusban kerülnek rögzitésre. Gyakran felmerül az az igény is, hogy a file-ok egymás között kommunikációt bonyolitsanak le.

4) Igen lényeges, hogy a file-kezelő rendszer fokozott *figyelmet forditson a hardware és software hibák automatikus felfedésére*, és a lehetőségekhez mért javitására. Ez vagy a *teljes file-rendszer duplikálásán*, vagy azon alapul, hogy a file-kezelő rendszer *időnként másodpéldányokat készit az összes*, de legalább az időközben *megváltoztatott file-okról*. Ezekkel a tartalékokkal a meghibásodott részek pótolhatók.

## *9.4. Az adatátviteli programcsomag*

Az *adatátviteli programcsomag* az adatkezelést végző *input-output vezérlő rendszer* szolgáltató képességeit bővitő rutinok összessége. Mielőtt tulajdonságait röviden megvizsgálnánk, néhány mondatban körvonalazzuk az input-output vezérlés általános kérdéseit.

Az input-output vezérlő rendszer nagyszámu, igen különböző, közeli és távoli input-output berendezés használatát teszi lehetővé a felhasználó számára. Eközben a berendezések müködtetésének és a felhasználók igényei kielégitésének olyan összehangolt optimalizálására törekszik, amelynek során a felhasználók hardware-függetlenitését is szem előtt tartja. Megvalósitásának alapelve, hogy a hardware berendezés-függő rutinokat hardware független, általános input-output rutinokkal illeszti a felhasználók felé.

A felhasználók a rendszer-software által értelmezett *input-output makroutasitásokon* és *logikai input-output rendszeren* keresztül informálják és vezérlik a fizikai input-output kezelő rendszert.

1) Az input-output adatforgalom az operativ tárban kijelölt *pufferterületeken* keresztül kerül kapcsolatba az operativ tárban lévő processzekkel. A pufferek mérete általában a fizikai rekordok méretének felel meg. A puffer illeszti a központi egységben futó processz és az input-output adatforgalom sebességkülönbségét, és teszi lehetővé a folyamatok szétválaszthatóságát. A pufferterület valamely processz részeként is definiálható, a processz és a puffer átmeneti függetlenithetősége szempontjából azonban előnyös, ha a puffert a processzen kivül helyezi el a rendszer-software. Egyetlen processz - soros információk hozzáférésére - külön az inputra és külön az outputra több pufferterületet átlapolva is használhat. Igy folyamatos müködése biztositható.

A pufferterületek kiosztása a hagyományos feldolgozásoknál többnyire statikusan, távadatfeldolgozási környezetben általában dinamikusan történik. A pufferek nyilvántartása tekintélyes rendszeradminisztrációval jár.

2) Az input-output adatforgalomban különösen gyakori, hogy a tranzakciós igények - legalábbis átmenetileg - sürübben lépnek fel, mint kiszolgálhatók lennének. Ilyen esetekben *várakozási sorokat* (queues) kell létrehozni, amelyekből valamilyen, *általában a prioritásokon alapuló algoritmus segitségével* választható ki a következő kiszolgálandó processz.

A várakozási sorok kezeléséhez szükséges információk az operativ tárban, mig maguk a várakozó tételek elsősorban háttértárakon helyezkednek el (ld. 9.1.).

3) Az input-output adatforgalomban igen jelentős szerepe van a megszakitásoknak. Az input-output berendezésektől érkező megszakitásokat *berendezés-függő interrupt rutinok* kezelik, amelyek felett egy *általános megszakitási rutin* áll. Az interruptok figyelmeztetik az input-output vezérlő rendszert a pufferek állapotváltozásaira és a várakozási sorok kiszolgálhatóságára.

4) Az input-output berendezéseket *ki kell osztani* a felhasználók között. Távadatfeldolgozási környezetben a kiosztást a dinamizmus jellemzi.

A berendezések egy része egyszerre csak egyetlen felhasználó számára elérhető, ezt *egyedileg használható* berendezésnek nevezzük.

A berendezések másik csoportja - ide a közvetlen hozzáférésü háttértárak tartoznak - *közösen használható*.

A tényleges kiosztás a berendezések használatának *megnyitásával* (opening) történik; majd ha a processz nem igényli tovább a berendezést, akkor "elereszti", használatát *lezárja* (closing).

A berendezések használatának megnyitása és lezárása a berendezésekhez rendelt, a berendezések vezérlését és állapotát leiró, hardware-függő *táblázatok* ellenőrzése és módositása segitségével történik. A táblázatok használata egyben biztositja a felhasználók, illetve processzek védelmét is egymással szemben.

Ehhez az input-output vezérlő rendszerhez illeszkedik az operációs rendszer részét képező *adatátviteli programcsomag*, a maga vezérlő rutinjaival, táblázataival, pufferterületeivel, várakozási soraival stb. (Pl. IBM: BTAM, QTAM, TCAM; SIEMENS: DUS, MCS.)

Az *adatátviteli programcsomag* feladata, hogy szimultán vezérelje:

- az adatátviteli hálózat és az üzeneteket felhasználó és feldolgozó programok között áramló, valamint
- az egyes hálózatágak között a feldolgozó központon az üzenetkapcsolás utján áramló

üzeneteket.

E feladatok a *hálózatvezérlés*, az *üzenet sorbaállitás* és az *üzenet szerkesztés* részfeladatokra bonthatók, melyek egymással szorosan összefonódnak.

Az adatátviteli vezérlő egység e feladatok jelentős részének végrehajtását magára vállalhatja, de a hardware vezérlő egységek általában csak elemi vezérlési funkciókat látnak el.

Az üzenetek utját a 17. ábra mutatja.

A továbbiakban csak az egyes részfeladatokhoz kapcsolódó legfontosabb funkciók felsorolására vállalkozhatunk.

a)·*Hálózatvezérlés*

A hálózat vonalait fizikailag az adatátviteli vezérlő egység vezérli a *csatornaprogram parancsai* alapján, mely folyamat során az üzeneteket kisérő *vezérlő karakterek* is a vonalra jutnak. Minden különböző vonalcsoporthoz, terminálcsoporthoz más-más vezérlési eljárás tartozik, amelyeket a rendszer *táblázatokban* rögzit.

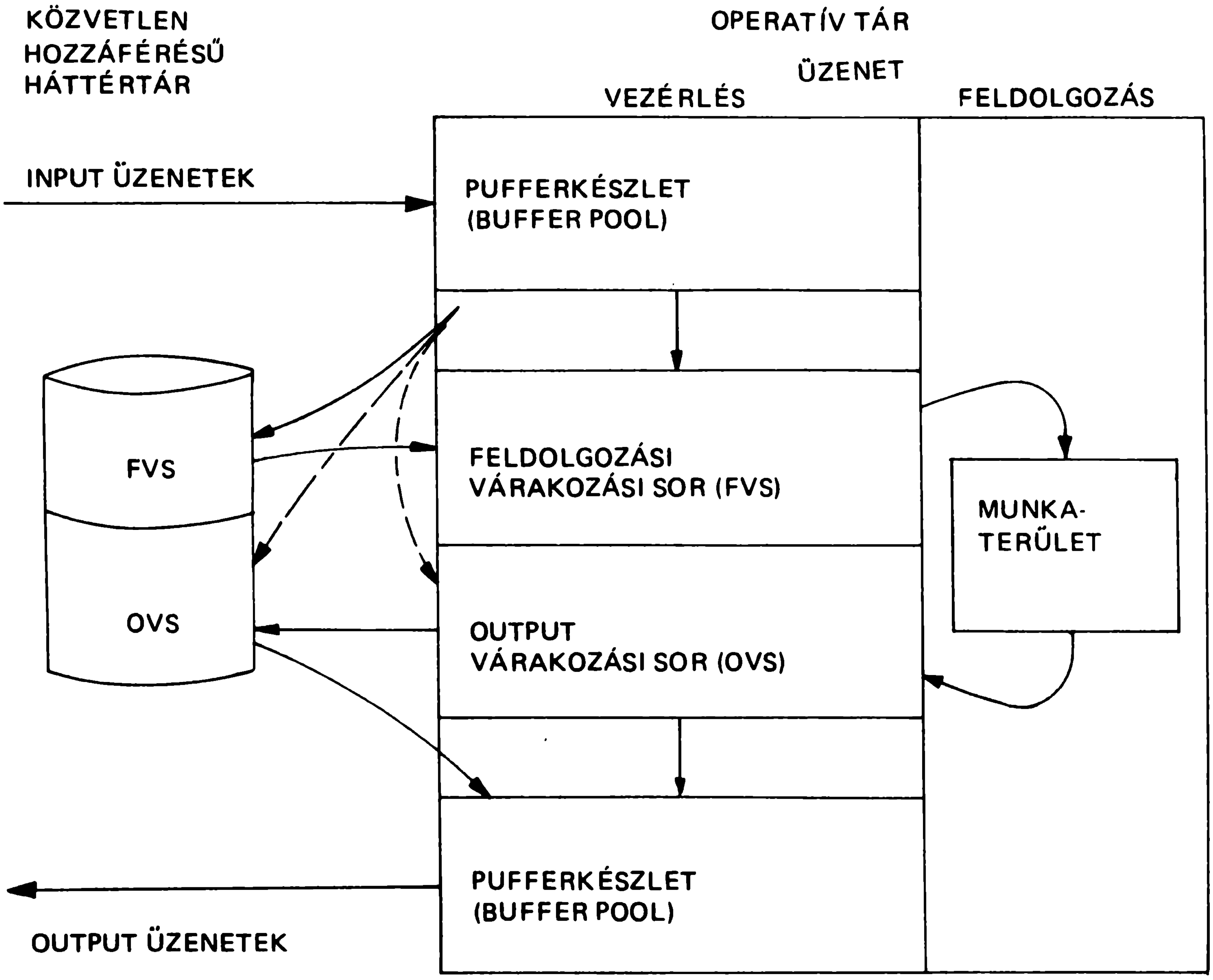
*A hálózatvezérlés kiválasztja a megfelelő eljárást*, és azt el is végezteti a *felhasználó makroutasitásai alapján* a rendszer által *generált csatornaprogram* segitségével. Megteremti a logikai és fizikai terminál cimek kapcsolatát és a logikailag kijelölt terminált kiválasztja, például polling vagy addressing utján. Figyelembe veszi az egyes terminálok specifikus vezérlési igényeit. Talán legfontosabb feladata az *üzenetekhez rendelt input-output pufferek kezelése*. A változó és megjósolhatatlan üzenethosszuság miatt általában egy sok elemi pufferből összeálló, nagy *pufferkészlet* (buffer pool) *elemi puffereinek dinamikus láncolásával és ujrafelszabaditásával osztja ki az aktiv vonalakhoz a puffereket*. Egyszerübb esetekben lineáris vagy gyürüs pufferelési stratégiát alkalmaz. Fontos feladata még a terminálok állapotának és az átviteli hibáknak az analizise és a szükséges beavatkozások megtétele az interruptok kezelése alapján.

b) *Üzenet sorbaállitás*

A feldolgozó központot elérő és elhagyó üzeneteket általában a processzorok leterhelésétől, az adatátviteli hálózat állapotától és az üzenetek sürgősségétől, prioritásától függő *várakozási sorokba* rendezi a rendszer.

Az üzenet sorbaállitás két irányu (ld. 17. ábra), egyrészt a feldolgozásra, másrészt a továbbitásra váró üzenetek képeznek sort. Rövid sorok az operativ tárban is képződhetnek, a hosszabb sorok háttértárakon helyezkednek el. Az üzenetküldő állomás az üzenet fejlécében meghatározhatja a várakozási sorba történő besorolás alapját képező üzenet-prioritást. A sorok kezelésére a 9.1. részben leirt elvek alapján kerül sor.

*Feldolgozás illetve továbbitás* előtt igen gyakran *át kell kódolni az üzeneteket*, ez kódtáblázatok segitségével történik és egyuttal a várakozási sorok átszervezését is jelentheti.


KÖZVETLEN
HOZZÁFÉRÉSŰ
HÁTTÉRTÁR
OPERATÍV TÁR
ÜZENET
VEZÉRLÉS
FELDOLGOZÁS
INPUT ÜZENETEK
PUFFERKÉSZLET
(BUFFER POOL)
FVS
FELDOLGOZÁSI
VÁRAKOZÁSI SOR (FVS)
MUNKA-
TERÜLET
OVS
OUTPUT
VÁRAKOZÁSI SOR (OVS)
PUFFERKÉSZLET
(BUFFER POOL)
OUTPUT ÜZENETEK

(A szaggatott vonallal rajzolt nyíl a csak üzenetkapcsolt üzenetek rövidített útvonalát jelzi.)

*17. ábra*
*Üzenetek utja a feldolgozó központban*

c) *Üzenet szerkesztés*

Az üzenet szerkesztő program a beérkező üzeneteket a rendszer felé uniformizálja, a kimenő üzeneteket pedig a rendeltetési hely igényei szerint átszerkeszti. Tipikus szerkesztési feladatok például a tömörités, illetve a tömöritett információ "felhigitása", az üzenetek "átszegmentálása", fejlécek, adatátviteli vezérlő, valamint terminál vezérlő karakterek analizise, beillesztése, elhagyása stb.

Az üzenetek átszerkesztéssel történő tömöritése előnyös, mind az adatátviteli sebesség, mind a tárolási tömörség szempontjából.

- . -

Az adatátviteli programcsomag müködtetéséhez mindig egy megadott minimális rendszer-konfiguráció szükséges.

A programcsomag moduláris felépitésü, éppen nem használt részei gyors háttértárakon helyezkednek el, de igy is nagy (10-20 Kbyte) operativ tárterületet igényel.

# 10.
# Adatátviteli kisszámítógépek alkalmazása

Mint az első fejezetben már emlitettük, az utóbbi néhány évben a kisszámitógépek - mint intelligens kisegitők - helyet kértek és nyertek a távadatfeldolgozó rendszerekben. Ez mindenekelőtt a fejlesztéshez és applikációhoz való rugalmas alkalmazkodó képességüknek és zuhanó áruknak köszönhető. /Irod. 2., 8., 9., 17./

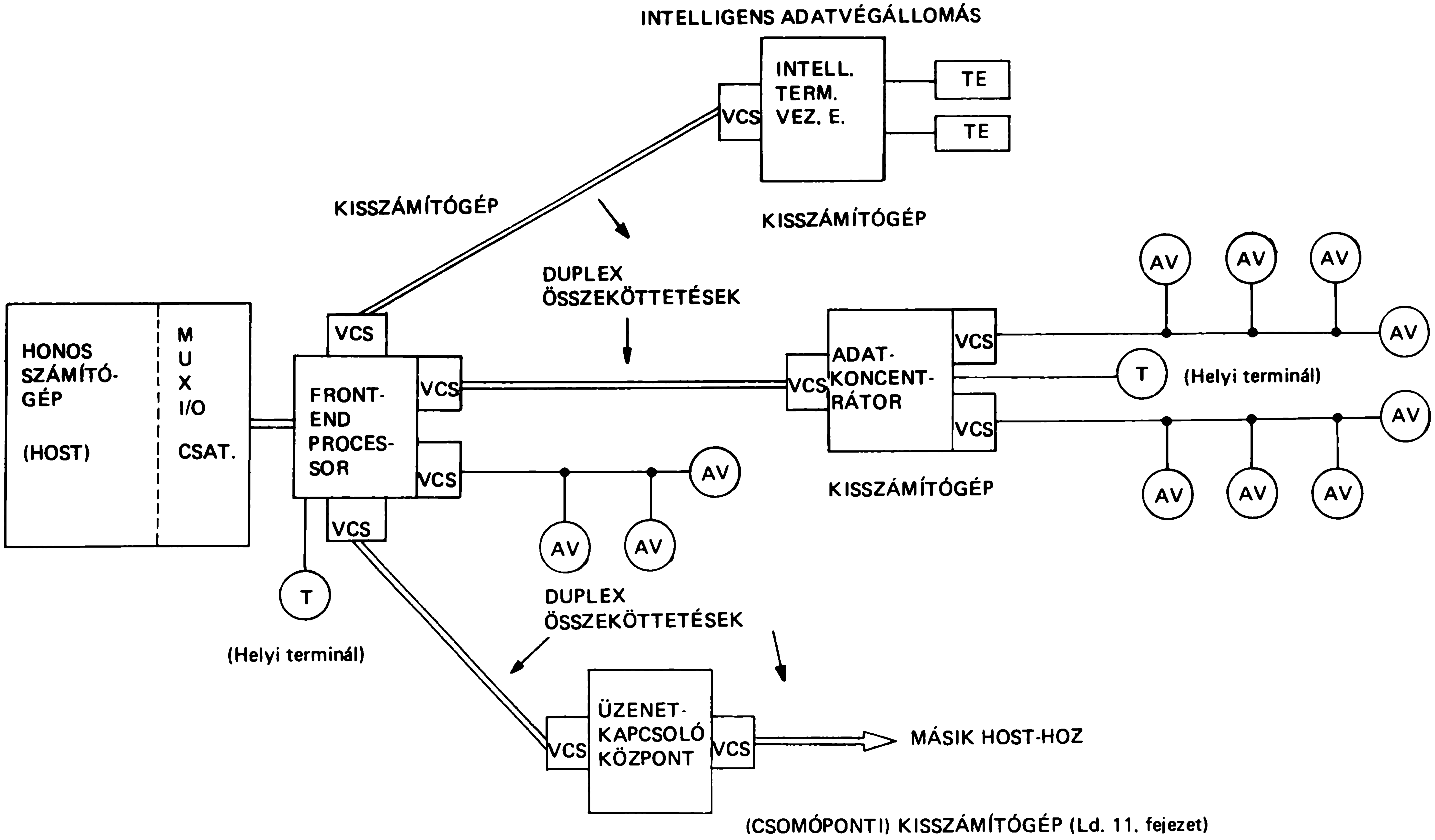
Bár az alkalmazott kisszámitógépek és az általuk ellátott feladatok skálája igen széles (gyakran mikro-, mini-, és midi computerekre bontva vizsgáljuk az adatátviteli számitógépeket) néhány, csaknem általánosnak tekinthető elvárás kiemelhető:

- az adatátviteli számitógép legyen hardware-ben és software-ben is moduláris felépitésü, bővithető;
- dolgozzon 8 vagy 16 bittel, de mindenképpen legyen képes 8 bites egységek (adatátvitelben szereplő karakterek, byte-ok) kezelésére, sőt esetleg egyszerü bitmanipulációkra;
- előnyös a mikroprogramozott vezérlésü központi egység;
- legyen többszintü megszakitási rendszere;
- az input-output rendszere rendelkezzen közvetlen tárhozzáféréssel (DMA: Direct Memory Access), lehetőleg mikroprogramozott csatornavezérléssel, nagygép csatornaillesztővel;
- legyen multiplex adatátviteli vezérlő egysége;
- a software követelmények már sokkal inkább alkalmazás-függőek, de minden esetben szükség van hatékony megszakitás kezelő- és puffertár kezelő rutinokra;
- kivánatos a fejlett hardware-software védelem;
- a vonalak és terminálok egyszerü kezeléséhez legyen makronyelve;
- legyen nagy megbizhatóságu és könnyen karbantartható.

Vizsgáljuk meg az alapvető alkalmazás-tipusok néhány tulajdonságát!

## *10.1. Programozható adatátviteli vezérlő egységek*

Az előző fejezetben az adatátviteli vezérlési funkciókat *logikailag* a központi számitógép operativ tárában elhelyezkedő, input-output csatornaprogram, *fizikailag* pedig egy hardware vezérlő egység végezte. A csatornaprogram-rendszer moduláris felépitésü, termináltipusonként, átviteltipusonként más-más parancssorozatot igényel, amelyek sok táblázatot használnak.


INTELLIGENS ADATVÉGÁLLOMÁS
INTELL. TERM. VEZ. E.
VCS
TE
TE
KISSZÁMÍTÓGÉP
KISSZÁMÍTÓGÉP
DUPLEX ÖSSZEKÖTTETÉSEK
HONOS SZÁMÍTÓ-GÉP
(HOST)
M U X I/O CSAT.
FRONT-END PROCES-SOR
VCS
VCS
VCS
VCS
ADAT-KONCENT-RÁTOR
VCS
VCS
VCS
T
(Helyi terminál)
AV
AV
AV
AV
AV
AV
AV
AV
AV
AV
AV
KISSZÁMÍTÓGÉP
T
(Helyi terminál)
DUPLEX ÖSSZEKÖTTETÉSEK
ÜZENET-KAPCSOLÓ KÖZPONT
VCS
VCS
MÁSIK HOST-HOZ
(CSOMÓPONTI) KISSZÁMÍTÓGÉP (Ld. 11. fejezet)

*18. ábra*
*Kisszámitógépek a távadatfeldolgozó rendszerben*

Ugyancsak moduláris felépitésü a hardware vezérlő egység, mely szintén terminál-tipusonként illetve az adatátvitel fajtája szerint sokféle vonalillesztő modult tartalmazhat. (Pl. IBM 2701, 2702, 2703, SIEMENS 4666.)

Ilyen megoldásnál az adatvonalak vezérléséhez nagy operativ tárterületre és bonyolult, költséges hardware berendezésre van szükség.

A különböző kódok közötti konverziót is a központi számitógép végzi el, ez viszont a központi egység idejének egy részét elrabolja a hasznos feldolgozástól.

Ugyancsak sok munkát igényel az üzenetek pufferelése, sorbaállitása és szerkesztése, valamint a különböző táblázatok kezelése.

Programozható adatátviteli vezérlő egységekkel ezeket a hátrányokat lényegesen csökkenthetjük.

A megoldást két irányban lehet keresni:

a) Programozható *adatátviteli processzort* alkalmazhatunk. Az adatátviteli processzor egyesiti magában az input-output csatorna és az adatátviteli vezérlő egység funkcióit.

Közvetlenül hozzáfér a központi számitógép operativ tárához. Emellett saját operativ tárral is rendelkezhet.

Az adatátviteli processzor speciális utasitáskészlettel, regiszterekkel, speciális hardware modulokkal van ellátva, "külön" programozható.

Ez a megoldás - bár a központi számitógép és a programozható adatátviteli vezérlés leghatékonyabb integrálása lenne - viszonylag kevéssé terjedt el, mivel a központi és adatátviteli processzorokat együtt kell tervezni. (Pl. SCC-6700 számitógép.)

b) A hardware adatátviteli vezérlő egység helyett - tehát egy I/O csatornához kapcsolva - alkalmazhatunk akár egy univerzális, akár egy speciálisan adatátvitelre tervezett programozható kisszámitógépet. Mindkét esetben a programozható adatátviteli vezérlő egység tehermentesiti a központi számitógépet az adatátvitellel kapcsolatos vezérlési és járulékos munkák nagy részétől mintegy előfeldolgozást végez. Innen kapta a *front-end processzor* nevet. Ez a megoldás egyre nagyobb jelentőségü, ezért röviden vizsgáljuk meg legfontosabb tulajdonságait. (Pl. UNIVAC C/SP; IBM 3705, SIEMENS 404/S).

Az ilyen front-end processzor jellegü adatátviteli vezérlő egységek a központi számitógép tehermentesitésének mértéke szerint három fokozatban csoportosithatók:

- emulátor front-end processzorok,
- ál front-end processzorok,
- valódi front-end processzorok.

1) *Emulátor front-end processzorok*

Emulátor front-end processzornak nevezzük azt a számitógépet, amely egy meglévő hardware adatátviteli vezérlő egység müködését, funkcióinak nagy részét hardware uton mikroprogramozás segitségével leutánozza, szimulálja.

Ez látszólag nem jelent előnyöket. A központi számitógépet igy nyilván nem tehermentesitjük. A gyakorlatban azonban mégis gyakran alkalmazzák ezt a megoldást, mint első lépést a valódi front-end processzor felé.

Alkalmazásának előnyei:

- az emulátor lényegesen olcsóbb lehet, mint az emulált célhardware;
- uj terminálok csatlakoztatása egyszerübb és olcsóbb;
- a központi számitógép adatátviteli programcsomagját nem kell megváltoztatni;
- továbbfejleszthető valódi front-end processzorrá.

2) *Ál front-end processzorok*

Az ál front-end processzor a hardware adatátviteli vezérlő egység emulálása mellett járulékos, korábban a központi számitógépben elvégzett feladatokat is ellát. Ilyen járulékos feladatok:

- terminálok közötti üzenetkapcsolás a központi számitógép beavatkozása nélkül,
- meghibásodott üzenetek automatikus, a központi számitógéptől függetlenitett ismétlése, illetve ismételtetése,
- üzenetek átmeneti tárolása a központi számitógép meghibásodása esetén stb.

Az ál front-end processzorok már a központi számitógépet is kismértékben tehermentesitik, például pufferterületek szabadulnak fel, a várakozási sorok rövidülnek, a megszakitások száma csökken.

3) *Valódi front-end processzorok*

A valódi front-end processzor a központi számitógép adatátviteli programcsomagja feladatainak nagy részét és a hardware adatátviteli vezérlő egység feladatait is átveszi. Alkalmazása tehát a központi számitógépben jelentős tár és processzor kapacitást szabadit fel.

A valódi front-end processzor az adatátviteli vonalak fizikai és logikai vezérlését is ellátja. Az eddig megismert front-end funkciók mellett feladatát képezi még:

- az adatátviteli vezérlő karakterek generálása és felismerése;
- önálló lekérdezés és kiválasztás;
- terminálkezelő algoritmusok generálása;
- az adatátviteli hibák kezelése;
- üzenetszerkesztés;
- üzenetpufferelés;
- kódkonverzió;
- üzenetek prioritáskezelése;
- üzenetek sorbaállitása saját, vagy a központi számitógéppel közösen használt közvetlen hozzáférésü háttértáron;
- egyszerü dialógus a terminálnál ülő operátorral.

Alkalmazásának egyetlen hátránya, hogy a központi számitógépben a rendszer-software lényeges változtatását igényli.

### *10.2. Intelligens terminálok*

A kisszámitógépre terminál vezérlési feladatok is bizhatók.

Intelligens terminál vezérlő egységek alkalmazása általában akkor célszerü, ha:

a) speciális, de egyszerü helyi feldolgozási igények lépnek fel az adatvégállomáson;

b) több egyszerü terminál egység közös vezérlését kell megoldani;

c) igen bonyolult terminál, például grafikus display vezérlési feladatait kell ellátni.

Az intelligens terminál vezérlő egység minden esetben részt vállal a terminál egységek és az adatátvitel vezérlésében, s emellett még járulékos, a központi számitógépet tehermentesitő feldolgozási munkákat is végezhet.

Mind interaktiv, mind remote batch terminálok esetében előnyös a vezérlés programozhatósága.

A legfontosabb intelligens terminál feladatok a következők:

- adatátviteli vezérlő bitek és karakterek fogadása, értelmezése és generálása;
- a terminál egységek logikai szétválasztása, cimzése, azonositása a feldolgozó központ polling-selecting eljárása szerint (lépcsős lekérdezés-kiválasztás);
- pufferterületek statikus vagy dinamikus kiosztása a terminál egységek között;
- a terminál egységek multiplex vezérlése;
- hibakorlátozás, automatikus ismétléskérés;
- automatikus válaszadás és hiváskezdeményezés;
- a feldolgozó központ és a terminál egységek kódjai közötti konverziók;
- üzenetszerkesztés, speciális jelek (pl. grafikus display vezérlő karakterek, APL nyelv speciális jelei) uniformizált üzenetekbe történő illesztése;
- formátumszerkesztés;
- adatkompresszió;
- helyi adatgyüjtés vezérlése;
- a terminál egységek közötti off-line tranzakciók lebonyolitása;
- rutinjellegü előfeldolgozás;
- saját file (pl. "mini mágneslemez") kezelési feladatainak ellátása;
- önálló, egyszerü dialógus lebonyolitása a terminálnál ülő felhasználóval stb.

E viszonylag általánosnak tekinthető feladatok mellett az egyes alkalmazásokhoz illeszkedő, speciális vezérlési funkciókat is elláthat az intelligens terminál vezérlő egység. Ilyen például a grafikus display ábráinak kicsinyitése, nagyitása, eltolása, elforgatása stb.

Mivel az intelligens terminál önálló helyi feldolgozásra képes, a feldolgozó központ vagy az odavezető összeköttetés meghibásodása vagy tulterheltsége esetén is találhat magának munkát és a terminálnál ülő felhasználóban is a folytonos foglalkoztatottság érzetét keltheti. Gyakori, hogy az intelligens terminál napközben előfeldolgozási, válogatási, rendezési munkákat végez, és csak éjjel-, amikor az adatvonalak terhelése és bérleti dija alacsonyabb - kommunikál a feldolgozó központtal.

## *10.3. Adatkoncentrátorok*

Ha több terminált egyszerü multiplexeléssel kapcsolunk egy közös adatvonalhoz, akkor a vonal kapacitása csak ugy használható jól ki, ha minden terminál egyszerre müködik. Igen sok gyakorlati esetben az a feltétel nem teljesithető, ezért a közös adatvonal kihasználatlan marad. Ha a terminálokról érkező, illetve az odatartó adatokat - az adatkoncentrálás (3.5.) elve szerint - *átmenetileg tároljuk*, majd *később továbbitjuk*, (store-and forward data concentration), akkor a közös adatvonal jól kihasználható lesz. A lassu, start-stop üzemü terminálokról összegyüjthetők az üzenetek és a nagysebességü közös vonal rövid ideig tartó "üzenetcsomók" formájában továbbithatja az egyes termináloktól származó üzeneteket. Ugyanakkor a nagy sebességü terminálok adatátviteli igénye is jól kielégithető, egyszerüen sürübben kapnak kiszolgálást, mint az alacsony sebességüek.

Igy lényegében tetszőleges terminálok együtt "üzenet-multiplexelhetők" és az üzenetek egy közös adatvonalra koncentrálhatók. Az ezt megvalósitó berendezést nevezzük *adatkoncentrátornak*. Az átmeneti tárolás következtében a közös adatvonal adatátviteli sebessége lényegesen kisebb lehet, mint a kiszolgált vonalak együttes adatátviteli sebessége.

Látható, hogy igen célszerü, ha az adatkoncentrátor rugalmasan tud alkalmazkodni a különböző terminálokhoz, ezért adatkoncentrátornak - bár célgépet is alkalmaznak - nagyon előnyös a programozható kisszámitógép.

Az adatkoncentrátor magára vállalhatja a terminál vezérlő egységek számos bonyolult feladatát, s ezáltal alkalmazása nem csupán az adathálózat terén, de a terminálokat illetően is jelentős megtakaritást jelenthet. Tulajdonképpen egyes bonyolult terminál vezérlő egységek is elláthatnak adatkoncentrálási feladatokat, ezért néha nem könnyü az adatkoncentrátor és a terminál vezérlő egység megkülönböztetése.

Általában azt mondhatjuk, hogy az adatkoncentrátorra az a jellemző, hogy a terminálokat további adatvonalakon, vonalcsatlakozókon át éri el.

Emellett az adatkoncentrátor hozzá közeli, helyi terminálokkal is foglalkozhat.

Az adatkoncentrátor a következő feladatokat láthatja el:

- Az aszinkron üzeneteket összegyüjti a termináloktól, a START és STOP jeleket elhagyja, és a közös nagy sebességü adatvonalon szinkron átvitel formájában továbbitja az üzenetet. A központi számitógéptől a start-stop terminálok felé beérkező üzenetekkel ugyanezt a müveletet ellenkező irányban végzi el.
- A beérkező üzenetekhez - általában dinamikusan, az üzenet tipusától, hosszától függő módon - pufferterületeket rendel. A pufferek vagy üzenet szegmensekhez vagy üzenetekhez tartoznak, tehát az adatkoncentrátor lényegében üzenet-multiplexelést végez. A lassu puffereletlen start-stop termináloknak megadja a lehetőséget, hogy az üzenetükhöz rendelt puffert speciális jelek segitségével az üzenet végéig lefoglalják, majd felszabaditsák (break).
- Egy puffert csak akkor tekint üresnek és szabadit fel, ha a belőle továbbitott üzenet hibátlan vételéről visszajelzést kapott, erről meggyőződött, tehát hibakorlátozást végezhet. A hibakorlátozás egyik gyakori fajtája, hogy a start-stop karakternyomtató terminál által küldött karaktereket egyenként visszaküldi, s ezek csak e "visszhang" hatására jelennek meg a papiron (echoing).
- Az adatkoncentrátor operativ tárában a beérkező és a továbbitandó üzeneteket is várakozási sorokba rendezi, képes blokkok vagy üzenetek összeláncolására.
- Az adatkoncentrátor képes lehet kódkonverzióra. Igy elérhető, hogy a központi számitógép minden üzenetet saját kódrendszerében kapjon meg, illetve továbbitson. Ez nagymértékben tehermentesiti a központi gépet. Ugyanakkor mód nyilik arra is, hogy a különböző müködésü és kódrendszerü terminálok a központi számitógép igénybevétele nélkül kommunikáljanak egymással.
- Ha több adatkoncentrátor csatlakozik a felfüzéses hálózatra, akkor az adatkoncentrátoroknak képesnek kell lenniük saját cimük felismerésére, a *polling és selecting folyamatban* való részvételre.
- Sok esetben az adatkoncentrátor terminál felőli oldalán is felfüzéses hálózatra csatlakozik. Ekkor *önállóan képes a polling és selecting vezérlésére*, a terminálok felé menő cimek kibocsátására, a terminálok válaszainak értékelésére.
- Az adatkoncentrátor *üzenetszerkesztési, formátumszerkesztési* feladatokat is elláthat. A központi számitógép felé menő nagysebességü adatvonal jobb kihasználása érdekében adattömöritést végezhet. Például egy display képernyőjén táblázatosan irt szövegben előforduló sok szóköz helyett csak egyetlen jelet és a szóközök számát továbbitja, a közös vonalon. Ezzel adatátviteli időt takarit meg.
- A felügyelete alatt dolgozó terminálok között a központi számitógép bevonása nélküli, távolsági üzenetkapcsolást vezérelhet.

- Az intelligens adatkoncentrátor a fentieken kivül természetesen tetszőleges előfeldolgozási feladatokat is elláthat, a felhasználó egyszerübb kérdéseit önállóan megválaszolhatja, a rendszer alkalmazásához illeszkedő speciális feladatokat végezhet el, és ezáltal nagymértékben tehermentesitheti a központi feldolgozó számitógépet.

# 11.

# A komplex számítástechnikai hálózatok néhány tulajdonsága

Mint az első fejezetben már röviden utaltunk rá, a komplex számitástechnikai hálózatokban önmagukban is szolgáltatásra képes, autonom, tipikusan egyközpontu távadatfeldolgozó rendszereket egyesitünk az egyes rendszerek autarchiáját feloldó közüzemi szolgáltató rendszerré. /Irod. 8., 9., 13., 19., 20./

Az ilyen rendszerek felhasználóinak arról sincs feltétlenül tudomásuk, hogy problémájuknak megoldása melyik feldolgozó központban, mely központok bevonásával történt. Ez csak ugy lehetséges, ha a rendszer logikai és fizikai vezérlése teljesen szétválik, az egyes számitóközpontok nyilvántartják egymás erőforrásait és interaktivan részesedhetnek azokban. Ahhoz, hogy a felhasználók rendelkezésére bocsátott erőforrások körét ilyen nagymértékben kiszélesitsük, biztositanunk kell, hogy a feldolgozó központok bármilyen párositásban, nagy megbizhatósággal és sebességgel, a kerülőutak választási lehetőségeit fenntartva, kommunikálhassanak egymással.

A feldolgozó központok fizikai kapcsolatát rendszerint ***nagysebességü, teljesen transzparens***, kisszámitógépek bevonásával realizált ***magasszintü kommunikációs hálózat*** teremti meg. (A feldolgozó központok közötti nagysebességü, - közvetlen vonalakkal vagy vonalkapcsolással realizált - hálózatot is magasszintü hálózatnak nevezhetik.)

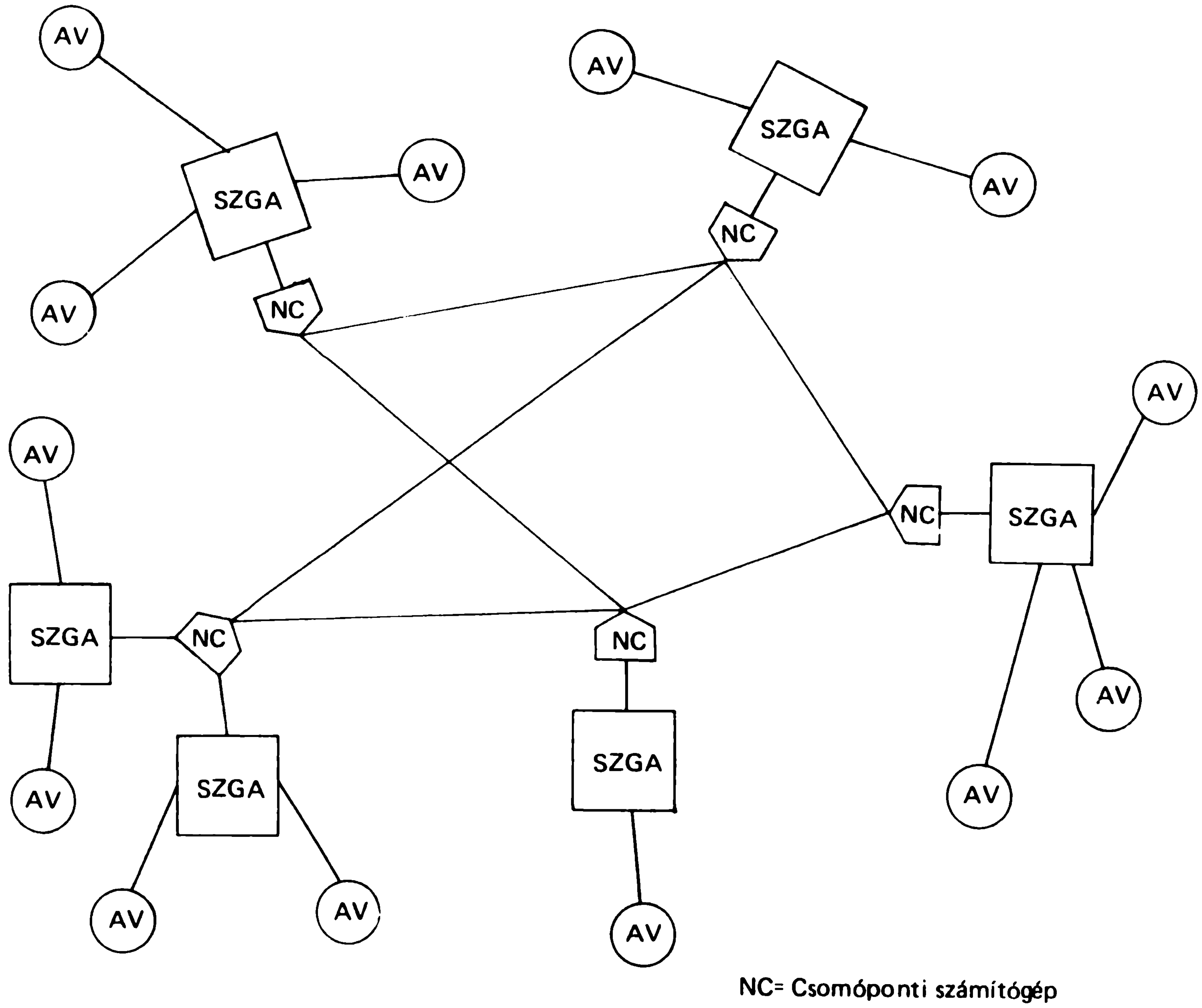
Ezek a kisszámitógépek a később vázolt feladatok ellátása mellett a feldolgozó központok adatátviteli processzorai, front-end processzorai is lehetnek.

A rendszer strukturája a 19. ábrán látható.

A magasszintü hálózat tipikusan rácsos szerkezetü, de általában nem teljes hurokhálózat. Fixen telepitett ágai ***duplex összeköttetést*** tesznek lehetővé. A hálózat csomópontjaiban adatátviteli vezérlési feladatok ellátására kitünően alkalmas, időazonos müködésü, ***csomóponti számitógépek*** helyezkednek el. A magasszintü hálózathoz az egyes feldolgozó központok mint "előfizetők" csatlakoznak. Egyetlen csomóponti számitógéphez egynél több feldolgozó központ is kapcsolódhat, akár lokálisan, akár távolról, bár az előbbi célszerübb és gyakoribb.

(Egyes megvalósitások megengedik, hogy a csomóponti számitógépekhez közönséges terminálokkal is csatlakozzanak felhasználók, ez azonban a csomóponttól járulékos funkciók ellátását igényli.)

A magasszintü hálózat az ***üzenetkapcsolás elvén*** müködik, ennek azonban speciális esetét, az ugynevezett ***üzenet-csomag kapcsolást***


AV
SZGA
NC
NC= Csomóponti számítógép

*19. ábra*
*Komplex számítástechnikai hálózat struktúrája*

(packet switching) valósitja meg. A hálózat belsejében haladó információt az jellemzi, hogy aránylag rövid, nagysebességü, más-más cimzetthez irányuló, időmultiplexelt blokkok formáját viseli. Az egyes összeköttetéseken hosszabb szünetek és csucsterhelések jelentkezhetnek. A kapcsolatfelépitési időknek viszont igen rövidnek kell lenni.

E nehézségek áthidalhatók az üzenetkapcsolás elvének alkalmazásával. Az egyes feldolgozó központok üzeneteinek többsége azonban - a feldolgozó központ és felhasználói kényelmét szolgálva - vagy tulságosan rövid, vagy tulságosan hosszu. Mivel minden üzenetet fejléccel és egyéb kisérő információkkal kell ellátni, tul rövid üzeneteknél az átvitel effektiv sebessége igen leromlana, tul hosszu üzeneteknél viszont tulságosan nagy tárterületre lenne szükség az egyes csomóponti számitógépekben az üzenetek átmeneti tárolására. Ezért a magasszintü hálózat ésszerüen választott méretü (például 1024 bit) és a könnyü kezelhetőség kedvéért *fix hosszuságu*, ugynevezett *üzenet-csomagokkal* dolgozik.

Egy ilyen üzenet-csomag felépitése lényegében megegyezik a 6.3. alfejezetben megismert keret-szerkezettel. A *fix-üzenethosszuság* igen megkönnyiti az *üzenet-csomag irányitási algoritmusok és stratégiák kidolgozását.*

A csomóponti számitógépek elsődleges feladata éppen az üzenet-csomagok irányitása. A feldolgozó központ és a csomóponti számitógép közötti interface-en a magasszintü hálózatba érkező üzeneteket üzenet-csomagokká kell alakitani. Ez elsősorban üzenet tördelésből áll, de előfordulhat, hogy több, azonos irányba tartó rövid üzenetből kell egyetlen üzenet-csomagot összeállitani. Az *üzenet-csomagok irányitása a fejlécük alapján történik*, a nyugtázást váró automatikus ismétlés kérés módszerével, a *csomópontok között folyó "párbeszéd" utján.* Minden küldő csomópont mindaddig tárol egy üzenet-csomagot, amig a szomszédos fogadó csomópont a helyes vételt nem nyugtázza. A nyugtázás egy speciális üzenet-csomaggal történik, amelyet sem a küldő, sem a fogadó csomópont nem őriz meg. Ezért előfordulhat, hogy egy nyugtázás elvész. Ilyen esetben a küldő csomóponti számitógép bizonyos türelmi idő elteltével megismétli az üzenet-csomagot és a fogadó-csomópont feladata a felesleges csomag-duplikáció elkerülése.

Az adatátviteli hibák felderitéséről hatékony ciklikus kód (pl. 24 bites) alkalmazásával, hardware CRC generálással és összehasonlitással gondoskodnak a csomóponti számitógépek. Átviteli hiba észlelésekor nemleges válasszal ismétlést kér a fogadó csomópont.

Az üzenet-csomagok vezérlésének másik alapvető mechanizmusa a csomópontok tulterhelését, "dugulását" akadályozza meg. Dugulás egyrészt a csomóponti számitógép véges tárkapacitása, másrészt az őt körülvevő csomópontok és hálózatágak meghibásodása következtében léphet fel.

A dugulás megakadályozására a feladott és a kommunikációs hálózatba belépő üzenetek hossza maximalizálva van.

A hálózat másik pontján belépő cimzett feldolgozó központ a teljes üzenet vételéről *készenléti nyugtázást* küld a feladó központnak, aki csak ez után adja a hálózatnak következő üzenetét.

A fentiekből látható, hogy a fizikai hálózatban az üzenetek üzenet-csomagokká alakitva, csomóponttól - csomópontig "araszolva" érik el céljukat.

Az egyes üzenet-csomag nyalábok csomópontonként átrendeződnek és gyorsan konvergálva haladnak a cimzett felé. Ezekről a manipulációkról még a feldolgozó központok sem szereznek tudomást. Az üzenetek logikailag ugyanugy hagyják el a feladó központot, mintha két központ közötti közvetlen üzenetkapcsolásról lenne szó.

Logikailag egy üzenettovábbitás két feldolgozó központ egy-egy processzének összekapcsolását jelenti. A processzek, illetve a hozzájuk tartozó felhasználók számára létrejön egy virtuális, *logikai hálózat*. E hálózat "csomópontjai" éppen a processzek és ezeket *logikai csatornák* kötik össze. Egy-egy logikai csatornát természetesen egy teljes üzenet átvitelének idejére fenn kell tartani, különben az összetartozó üzenet-csomagok nem lennének azonosithatók. A logikai csatornák fizikailag dinamikusan változva realizálódnak, mivel csak a magasszintü hálózattól függ, hogy melyik üzenet-csomagot milyen fizikai utvonalon juttatja el a cimzett központba. Egy adott időpillanatban több, független logikai csatorna létezik a rendszerben. Ezek megkülönböztetésére minden *üzenet-csomag fejléce tartalmazza a feladó és a cimzett azonositója mellett a hozzárendelt logikai csatorna nevét*, valamint több csomagra bontott üzenetek esetében a *csomagok sorszámát is*. A felhasználók nemhogy a magasszintü hálózat eseményeit, de még a feldolgozó központok közötti üzenettovábbitást sem érzékelik.

De térjünk vissza a magasszintü fizikai hálózatban lezajló utirány-kijelölés problémaköréhez.

Nyilvánvaló igény: a magasszintü hálózatnak arra kell törekednie, hogy az átlagos üzenet-csomag továbbitási idő minél rövidebb legyen. Ezt ugy kell valamilyen *utvonal döntési stratégia* szerint optimalizálnia, hogy közben figyelembe veszi a:

- forgalom-eloszlás változásait,
- az üzenetek prioritását,
- a topológia adta lehetőségeket,
- a hálózatágak és csomópontok meghibásodását stb.

Rögzitett utvonal kijelölése kedvező lenne ugyan nagy tömegü információ továbbitására, de igen érzékeny lenne az utvonal ágainak és csomópontjainak letörésére, ezért a magasszintü hálózat általában rendelkezik a *dinamikus és adaptiv utvonalválasztás* lehetőségével.

A csomóponti számitógépek az *utvonal kijelölését* egyrészt az üzenet-csomagok *fejlécében* foglalt információk, másrészt a hálózatról saját maguk által kialakitott *térkép* alapján végzik. Az irányikijelölési procedurát minden egyes üzenet-csomagnál megismétlik, ez teszi lehetővé a hálózat és a forgalom változásainak dinamikus figyelembevételét.

Iránykijelölésnél - a fenti tényezőket *analizáló döntési stratégia* eredményeként - egy csomóponti számitógépben a cimzett azonositóját és egy vagy több szomszédos csomópontba vezető, egy vagy több csatorna sorszámát összerendelő, *kapcsoló vektort* generál a rendszer. A kapcsoló vektorokat befolyásoló tényezők analizise elvégezhető egy - a többi csomópont lekérdezésének képességével rendelkező - *kitüntetett csomóponti számitógépben* is, de hatékonyabb, ha *minden csomóponti számitógép* maga végzi el *környezetének "feltérképezését"* és kapcsoló vektorainak beállitását.

E rövid fejezet csupán megközelitő leirása annak, hogy a komplex számitástechnikai hálózatokban milyen nehézségek lépnek fel a feldolgozó központok információcseréje során.

Ezek mellett számos lényeges kérdést kell még megoldani, amelyekkel itt nem foglalkozhattunk. De talán érzékelhető, hogy ezen az uton haladva, olyan komplex információs rendszerek valósithatók meg, amelyekben mind az egyes feldolgozó központok és a közöttük kifeszitett hálózat terheléselosztása, mind pedig akár a távoli feldolgozandó adatoknak, akár a távoli feldolgozó programoknak "helybe" hozása, távoli file-ok közvetlen elérése lehetséges.

# Irodalomjegyzék

/ 1/ Wright, P.: Facing a Booming Demand for Networks, Datamation, Nov.1973.

/ 2/ Infotech ed.: Computing Terminals, Infotech 1972, Maidenhead.

/ 3/ Hopewell, L.: Trends in Data Communications, Datamation, Aug. 1973.

/ 4/ OMFB: Számitógépek alkalmazása; Adatátviteli hálózat kérdései, OMFB 16-7001-Et, 1971. nov. Budapest.

/ 5/ Martin, J.: Telecommunications and the Computer, Prentice-Hall Inc. 1969. Englewood Cliffs, N.J.

/ 6/ Siemens: Siemens-Datenfernverarbeitung, SIEMENS AG. 1970. München.

/ 7/ Mazgon, S.: Adatátvitel a Magyar Posta vonalain, SZÁMOK, Távadatfeldolgozás c. speciális tanfolyam anyaga, 1973. Budapest.

/ 8/ Infotech ed.: Computer-based Data Communication Networks. The Infotech Professional Development Programme Lecture Manual, 1973. London.

/ 9/ Martin, J.: Teleprocessing Network Organization, Prentice-Hall Inc. 1970. Englewood Cliffs, N.J.

/10/ Martin, J.: Design of Real-Time Computer Systems, Prentice-Hall Inc. 1967. Englewood Cliffs, N.J.

/11/ Csomor, Gy.: Limperger, I., Lohonyai, M.: A távfeldolgozás alapjai. SZÁMOK tankönyv, előkészületben.

/12/ Dr.Gordos, G.: Információ- és kódelméleti alapok, SZÁMOK, Távadatfeldolgozás c. speciális tanfolyam anyaga, 1972. Budapest.

/13/ Martin, J.: Systems Analysis for Data Transmission, Prentice-Hall Inc. 1972. Englewood Cliffs, N.J.

/14/ Martin, J.: Designs of Man-Computers Dialogues, Prentice-Hall Inc. 1973. Englewood Cliffs, N.J.

/15/ Martin, J.: Programming Real-Time Computer Systems, Prentice-Hall Inc. 1965. Englewood Cliffs, N.J.

/16/ Watson, R.: Time-sharing System Design Concepts, McGraw-Hill, Inc. 1970. New York.

/17/ Schoeffler, J.; Temple, R.: Minicomputers: Hardware, Software and Applications, IEEE PRESS, 1972, New York.

/18/ Hirsch, P.: EPPS: Is There a Market for Packet Networks, Datamation, Febr. 1973.

/19/ Németh, J.: Számitógép hálózatok müködtető rendszerei, SZÁMOK, Távadatfeldolgozás c. speciális tanfolyam anyaga, 1972, Balatonfüred.

/20/ Wollner, R.: Számitógép hálózatok rendszertechnikája, SZÁMOK, Távadatfeldolgozás c. speciális tanfolyam anyaga, 1972. Budapest

HUBA ZOLTÁN

# Az időosztásos üzemmód

LEKTORÁLTA:
SÁROSSY JÓZSEF

# Bevezetés

Az időosztásos rendszereket a párbeszédes üzemmódu számitógéphasználat igénye hivta életre. Ennek a géphasználati módnak a gazdaságos megteremtése néhány speciális problémának a megoldását követelte. Az erre irányuló kutatás néhány, merőben uj hardware, illetve software ujdonságot produkált. Az itt következő tanulmány az időosztásos rendszerek speciális problémáit és ezek megoldásának elveit tárgyalja.

Az időosztásos rendszerek fejlesztése során feltárt ujdonságok jórészt a nem időosztásos rendszereknél is haszonnal alkalmazhatók. A legkülönfélébb rendszerek alkalmazzák pl. a reentrant software vagy a virtuális tár konkrét megvalósitását. Ezek a dolgok ma már a számitástechnika általános fegyvertárához tartoznak, és ezért részletes ismertetésük jelen sorozat későbbi köteteiben megtalálható lesz. Ennek a tanulmánynak ezért nem célja a konkrét megvalósitás különböző formáit ismertetni, hanem azt megmutatni, hogy az időosztásos rendszerek mely problémáinak megoldására születtek, és hogyan szolgálják a megoldást. A tanulmány tehát összefoglaló, illetve rendszerező céllal készült és ezért került a sorozat első kötetébe.

A szerző

# 1.
# Bevezetés az általános célú időosztásos rendszerekbe

## *1.1. Az időosztásos üzemmód kialakulásának okai és az üzemmód definiciója*

### *1.1.1. Az időosztásos üzemmód kialakulása*

Mikor közel 30 évvel ezelőtt, 1946-ban, majd három évi kutató-fejlesztő munka után Prosper Eckert és John Mauchly az első számitógéphez ültek, hogy egy matematikai probléma megoldására programot készitsenek, aligha kellett a számitógép hatékony felhasználásával törődniök. Bár a világ első elektronikus számitógépét, az ENIAC-ot (=Electronic Numerical Integrator and Computer) hamarosan ujabb számitógépek követték, a számitógép az első időben matematikusok, mérnökök kutatási eszköze maradt. Minthogy nagyon kevesek rendelkeztek elegendő tudással ahhoz, hogy egy számitógépes programot irhassanak, ezek a személyek nyugodtan kisajátithatták az óriási gép idejét, s a vezérlőpultnál ülve, lépésről-lépésre próbálva, javithatták programjaikat. A számitógép ekkor még csak technikai részből (hardware) állt. A hardware fejlesztői igyekeztek minél több segitséget nyujtani a programok próbájához - a számitógép vezérlőpultja tele lett jelzőlámpákkal, nyomógombokkal és számjegyes kijelzőmezőkkel. Ezzel a számitógép fejlődésének hőskora - a"*hardware évtizede*"- le is zárult.

A számitógépes technika rohamosan terjedt; egyre nagyobb teljesitményüvé és gyorsabbá váltak ezek az igen költséges berendezések, és egyre több ember ismerkedett meg a használatukkal. A ráforditások minél gyorsabb megtérülése érdekében előtérbe került a számitógép hatékonyságának növelése.

Ezt próbálták megoldani a hardware fejlesztők törekvései (tranzisztorok, megszakitási rendszer stb.), és a programozók egy részének erőfeszitései is. Ez utóbbiak kidolgozták az adott gép teljesitőképességének lehető legnagyobbmérvü kiaknázását szolgáló programelemeket, melyeket használatra felajánlottak programozó társaiknak. Ezen elemek használatát hamarosan kötelezővé is tették, ezzel megjelent a számitógép "láthatatlan fele", a software is. A softwarefejlesztők célja a maximális hatékonyság elérése volt, ennek utja pedig az, hogy minden emberi gondolkodást igénylő funkciót eltávolitottak a számitógép közeléből. Az egyes programok futtatásához szükséges adatokat jóelőre kártyába kellett lyukasztani, a program által igényelt adattárakat előkésziteni, és a programokat - egyiket a másik után - pergő sietséggel futtatni. A számitógépet tehát nem feladatonként inditották el, hanem a feladatokból képzett feladatsort, köteget egyvégtében végeztették el. Ezt a géphasználati formát ez utóbbi okból nevezzük kötegelt feldolgozásnak (batch processing). A bonyolult vezérlőpult eltünt a gép előteréből, valamint eltünt a programozó is, maradt az operátor és egy irógép, melyen a software állandóan memóriában tartózkodó (rezidens) része, a supervisor (vagy monitor, máshol executiv) hajszolta környezetét a minél pergőbb előkészitésre.

A géphez való hozzáférés a számitógép fejlődésének második szakaszában - a *"software évtizedében"* - igen problematikussá vált. A programozó programját megirta, kártyába lyukasztotta, majd leadta a gépteremnek. A gép kiszolgáló személyzete ezt a feladatot beillesztette egy végrehajtásra váró feladatkötegbe, majd leforditotta. A forditóprogram által produkált hibajelzéseket a programozó visszakapta, ennek alapján programját módositotta, majd ujra leadta forditani,... Hogy ez a helyzet hamarosan nem vált a programozó számára reménytelenné, az csak az időközben óriásit fejlődött programnyelvi lehetőségeknek, ill. a bonyolult programpróbálási segédleteknek (debugging aids) volt köszönhető. Mindenesetre azok a felhasználók, akik programjaikat igen gyakran módositják, módositás után az uj változatot futtatják, a futtatás eredményét elemzik, összevetik kisérleti tapasztalataikkal, majd ujra módositanak stb... hamar kiszorultak a gépteremből. Pedig ezek a felhasználók - a tudósok és kutatók - igénylik a számitógépet. Ezeknek kielégitésére alakult ki egy uj számitógéphasználati rend.

Ezen uj rend visszaadja a teljes számitógépet a felhasználónak, aki ugy használja azt, mint annak idején a számitógép bölcsőjénél tevékenykedő kutatók tették, ugy biztositva a drága berendezés teljesitőképességének maradéktalan hasznositását, hogy igen sok felhasználóval teszi ugyanezt. Természetesen ez a sok felhasználó nem helyezhető el a gépteremben.

A távfeldolgozás fejlődése lehetővé tette, hogy a különböző felhasználók irodájába, laboratóriumába vigyük egy számitógép óriási logikai és aritmetikai müveletvégző kapacitását. Igy tette lehetővé, hogy a kutató ember visszakapja a számitógépet a fejlődés harmadik szakasza, - a *"távfeldolgozás évtizede"*.

### *1.1.2. Az időosztásos üzemmód definiciója*

Az időosztásos üzemmódban (time sharing) dolgozó számitógép felosztja idejét a különböző felhasználók között. Az időosztás (time sharing) fogalmát ilyen értelemben széles körben használják. Mondhatjuk például, hogy a különböző input/output csatornák időosztásos módban használják a központi memóriát stb.

A számitógéphasználat időosztásos üzemmódjánál azonban nem ez a lényeg.

*Általános célu időosztásos rendszernek* az olyan ***használati üzemmódban*** müködő számitógépes rendszert nevezzük, mely lehetővé teszi, hogy a rendszerhez kapcsolt ***bármelyik felhasználó, bármilyen időpillanatban teljes*** számitógépes munkaválaszték felett rendelkezzen.

A teljes számitógépes munkaválaszték elemei:

program irása, módositása, próbája, futtatása;
adattárak létesitése, törlése, módositása, kiegészitése.

Nem nevezzük tehát általános célu időosztásos rendszernek a mai modern kötegelt feldolgozást végző rendszereket, bár egyidőben több programot hajtanak végre (multiprogramozás), és a rendszer idejét felosztja az egyes programok között, mert az egyes felhasználók rendszerhez történő hozzáférése időben nem tetszésszerinti, bár teljes mértékü.

Nem nevezzük általános célu időosztásos rendszernek a mai modern reálidős (real time) feldolgozási módban müködő rendszereket sem, bár ott bármelyik felhasználó, bármilyen időpontban információt cserélhet a számitógépes rendszerrel, mert nincs lehetőségük rendelkezni a teljes munkaválaszték felett. E rendszernek legelterjedtebb formája ma a repülőgéptársaságok számitógépes helyfoglalási rendszere. Ez esetben az adatátvitel segitségével a rendszerhez kapcsolt végállomásnál (terminál) ülő felhasználó csak egy központi adattárat kérdezhet le vagy módosithat, többre nincs lehetősége. Mindenesetre nyilvánvaló, hogy első pillantásra a reálidős (real time) rendszer igen hasonlit az általános célu időosztásos rendszerhez. Mindkettőnél a gép üzemét irányitó software-nak, - az operációs rendszernek - kell megoldania, hogy a terminálnál ülő felhasználónak ugy tünjön, mintha a számitógéprendszert csak ő használná. A számitógépes erőforrások pillanatról-pillanatra változó (dinamikus) szétosztása (allokációja) a felhasználó szempontjából észrevehetetlen, átlátszó kell legyen. Az igen bonyolult feladat végrehajtásánál a reálidős rendszer operációs software-ja könnyebb helyzetben van, mert pontosabban ismeri a felhasználó igényének kielégitéséhez szükséges erőforrásokat, bár nem tudja, hogy ezek az igények milyen eloszlásban és milyen időpontban lépnek fel, és a felhasználó kiszolgálására engedélyezett időintervallum (válaszidő) általában szükebb, mint az időosztásos rendszereknél. Az előbbiekből érthető, hogy miért van az, hogy sokan a reálidős üzemmódot az időosztásos üzemmód egy fajtájaként tartják számon, azonban a felhasználó *teljes* kiszolgálására való törekvés olyan többlet, amely olyan uj hardware, illetve software elvek implementálását igényli, amely szükségtelen egy reálidős rendszernél, de mely miatt az időosztásos üzemmód nem tekinthető a multiprogramozott kötegelt feldolgozás továbbfejlesztésének sem, hanem merőben uj üzemmódként fogható fel.

## *1.2. Az időosztásos rendszerek jellemzői és forrásai*

### *1.2.1. Az időosztásos rendszer és az ember*

A számitógép fejlődése - a software évtizedében - elsősorban a gép igényeit, jellemzőit tartotta szem előtt, és kevésbé törődött az emberrel. Az időosztásos rendszerek fejlesztői pedig elsősorban a felhasználó embert, annak kényelmét tartják szem előtt. Az állitás bizonyitására állitsuk szembe a mai modern kötegelt feldolgozási (batch processing) rendszereket az időosztásos rendszerekkel (time sharing) (1. táblázat).

1. táblázat

BATCH PROCESSING TIME SHARING
üzemmódu rendszer és a külvilág kapcsolatának
jellemzői

| | |
|---|---|
| meglehetősen rugalmatlan futtatási utasitásrendszer | lehetőség arra, hogy a futtatási utasitást menet közben - a hibáknak és eredményeknek megfelelően - változtassuk |
| gyors, előre elkészitett input | lassu, kézi input |
| igen gyors és nagytömegü output | lassu, kismennyiségü output |
| programjavitás géptől távol, a programozó idejében | programjavitás egyszerüen és gyorsan, legtöbbször akár a futás idejében is |
| a rendszer üzenetei rövidek, melyek kézikönyv segitségével fejthetők meg | általában kézikönyvre nincs szükség, mig az ember a terminálnál ül. |

További bizonyitékként néhány modern időosztásos rendszer egy-két olyan szolgáltatását ismertetjük, amely a használat kényelmét fokozza.

Az időosztásos üzemmódu rendszer igyekszik lehetővé tenni, hogy szolgáltatásait a számitógépes munka szempontjából képzetlen, vagy valamikor kiképzett, de már sok mindent elfelejtett felhasználók is élvezhessék. Ennek érdekében egyes rendszerek a felhasználó kérésére áttekintést adnak szolgáltatásaikról (System Information Retrieval). Például: a terminálnál helyetfoglaló felhasználó a HELP (=segits) rendszerutasitás segitségével kapcsolatba kerül a fentemlitett szolgáltatással. A rendszer információs modulja a

TEACH? (=tanitsalak? )

üzenettel érdeklődik,

hogy teljes áttekintést kell-e adni, vagy egy konkrét eset magyarázatát. Ha a felhasználó erre az üzenetre

YES-szel (=igen)

válaszol, akkor a modul a terminálon keresztül ismerteti vele az összes rendszerparancsot. Gyakorlottabb felhasználó az információs modul utasitásait használva, egy speciális utasitás részletes magyarázatát kérheti. Ez a részletes ismertetés kiterjed az utasitás formájára, funkciójára, vagy az utasitás használata során kapott hibajelzések részletes ismertetésére.

Egyes időosztásos rendszerek igen udvarias szolgáltatása, hogy igyekeznek megismerni a konkrét felhasználók szokásait és ezekhez alkalmazkodni. Például: a forditóprogramoknak különböző szolgáltatásai vannak, melyeket a felhasználónak kell megadnia. Az egyes felhasználók általában megszoknak egy kombinációt, s a forditóprogramot mindig ezzel a szolgáltatással használják. Annak érdekében, hogy a felhasználónak ne kelljen óhaját minden programforditási menetnél megismételni, a rendszer megőrzi ezt az információt, s csak akkor kell ujra explicit módon ezeket megadni, ha a szokástól eltérő kombinációt óhajt definiálni.

A modern rendszerek az egyes felhasználókról vezetett feljegyzéseikben (User Directory) nagyon sok ehhez hasonló információt őriznek. Érdekes speciális szolgáltatás a felhasználók közötti levelezés lehetővé tétele (Inter-user Mail Capability). E szolgáltatás keretében az egyes felhasználók üzenetet hagyhatnak bármelyik másik felhasználó számára. Amig a cimzett nincs kapcsolatban a rendszerrel, illetve, amig nem érdeklődik a rendszernél, hogy nincs-e számára üzenet, addig ez azt megőrzi. Lehetőség van arra is, hogy egy vállalat összes leányvállalata (az összes hozzá kapcsolódó terminál számára) adjon fel üzenetet, vagy a rendszer az összes felhasználónak (összes terminál számára) tároljon valamilyen közérdekü üzenetet. Természetesen megfelelő megállapodás esetén két terminál között tetszésszerinti levelezést is le lehet folytatni ennek a szolgáltatásnak a segitségével.

### *1.2.2. Az időosztásos rendszerek formái*

Az 1.1.1. pontban megismertük, hogy az időosztásos üzemmódu rendszerek történelmileg alakultak ki, az addigi üzemmódokat lehetővé tevő operációs rendszerekből, azok helyett. Ebből következik, hogy tiszta időosztásos rendszer kevés van, annál több olyan, amely egy más tipusu üzemmódból igyekszik az időosztásos üzemmód jellemző szolgáltatásait nyujtani. Eltekintve ezektől az időosztásszerü, illetve ál-időosztásos rendszerektől, az alábbi rendszerformák ismerhetők fel.

### *1.2.2.1. A tulajdonképpeni általános célu időosztásos rendszer*

Egy erre a célra felépitett időosztásos rendszer a technika mai állása mellett lehetővé tudja tenni, hogy egy nagy számitógép egyidőben 20, 50, 100, sőt még több felhasználót tud egyidőben kiszolgálni. Termináljától függő módon, a felhasználó a rendszerrel állandó kapcsolatot tud fenntartani, sokféle rendszerfunkciót tud használni: programot irni, javitani, futtatni, a futást megszakitani, vagy valamilyen feltétel felléptekor (pl. ha egy változó értéke tullépi a megadott értéket) megszakittatni, a programot módositani és továbbfuttatni a megszakitási állapottól.

A felhasználónak lehetősége van file-okat képezni, azokat egyedül vagy más felhasználókkal közösen használni, módositani, vagy törölni. Mindezen funkciókat a felhasználó ugy birtokolja, mintha egyedül ő használná a számitógépet. Az a hatalmas erőfeszités, amit az operációs rendszer a sokféle tevékenység gördülékeny lebonyolitása érdekében végez, észrevehetetlen kell hogy maradjon számára.

Az általános célu időosztásos rendszer iskolapéldája az MIT által kidolgozott MULTICS (*Mult*iplexed *I*nformation and *C*omputing *S*ervice), valamint a System Development Corporation Q-32, illetve a Berkeley egyetemen kifejlesztett XDS-940 rendszer.

### *1.2.2.2. Időosztásos tevékenységet lehetővé tevő multiprogramozható kötegelt feldolgozási (batch) rendszerek*

E rendszereknél amellett, hogy néhány (10-50) felhasználó a rendszerrel állandó kapcsolatot tud fenntartani, a közismert kötegelt módu feldolgozás is folyik. Természetesen az utóbbi a szokásos módon, a gépteremben elhelyezett perifériák segitségével bonyolódik, s az ilyen tevékenységet a gépteremben elhelyezett vezérlőirógépről kell inditani, bár néhány rendszer lehetővé teszi azt is, hogy a termináloknál ülő felhasználók is inditsanak kötegelt feldolgozásu munkákat (batch job). Ezeknél a rendszereknél a termináloknál ülő felhasználók munkájuk közben használhatják a rendszernek általában kötegelt munkákhoz használt perifériáit is. Vagyis lehetőség van arra, hogy például egy terminálról irt és futtatott program futása közben sornyomtatóra vagy mágnesszalagra irjon. Ezek után az olvasó azt a következtetést vonhatja le, hogy tulajdonképpen a párbeszédes (interaktiv) kötegelt feldolgozási (batch) rendszerek tökéletesebbek, mint az általános célu időosztásos (time sharing) rendszerek. Ez abból a szempontból igaz, hogy egy ilyen rendszer több lehetőséget ad (pl. kötegelt feldolgozás inditása), de ezért azzal kell fizetni, hogy a rendszer egyidőben sokkal kevesebb vonalban lévő (on-line) felhasználót tud kiszolgálni, mint ezt a hardware mérete és egy tökéletesebb erőforrás-elosztó algoritmussal müködő software vezérlete lehetővé tenné. Ezen rendszereknél ugyanis a kötegelt módu feldolgozás optimális lebonyolitását tüzik ki fő célként.

Ezen tulmenően az ilyen rendszerekkel állandó kapcsolatot fenntartó (interaktiv) felhasználóknak még néhány, a programok irására, futtatására, a file-ok kezelésére vonatkozó megkötéssel is számolniuk kell.

Ilyen kettős használatu operációs rendszer az IBM OS/360, valamint a UNIVAC EXEC 8, illetve Honeywell - Bull - General - Electric GECOS RENDSZER.

### *1.2.2.3. Speciális időosztásos rendszerek*

Az időosztásos üzemmód ma már korántsem az óriás számitógépek privilégiuma, jónéhány kisszámitógépet gyártó cég is fejlesztett ki időosztásos üzemmódu operációs rendszert. A kisszámitógép korlátozott lehetőségei nem teszik célszerüvé általános célu időosztásos rendszer implementálását. Egyrészről ez az oka a speciális célu időosztásos rendszerek kialakulásának. Ezek a rendszerek egyetlen interaktiv programnyelven történő programozást (pl. APL vagy SYMBOL) és a program futtatását teszik lehetővé. Ezek az un. egynyelvü rendszerek általában 10-20 terminál egyidejü kezelését tudják ellátni.

Másrészről, különösen az időosztásos rendszerek kialakulásának első fázisában, igen sok időosztásos rendszer magán viseli annak nyomát, hogy kötegelt feldolgozási rendszerből fejlődött ki. Ezeknél a rendszereknél a termináloknál ülő felhasználó kapcsolattartása (interaktivitása) a rendszerrel korlátozott, a program irása nehézkesebb, (mint erre a 3. fejezetben részletesen visszatérünk), a file-ok használatában sok a kikötés.

## *1.3. Az időosztásos rendszerek architekturája*

### *1.3.1. Lényegi és technológiai problémák*

Mint ezt Saltzer /Irod. 1./ ajánlotta, célszerü a problémákat két kategóriába osztani. Az első kategóriába tartoznak a lényegi problémák (intrinsic), a másodikba pedig a technológiaik. A lényegi problémák a rendszer céljával kapcsolatosak. Például: a terminálnál ülő felhasználók adattárakkal rendelkeznek, ezeket módositják, programokat irnak stb. Ehhez meg kell oldani, egyrészt, hogy lehetőségük legyen elérni, módositani, megosztani a rendszer által őrzött információt, másrészt azonban ezt az információt meg kell védeni az illetéktelen, illetve hibásan dolgozó felhasználóktól, sőt a rendszerhibáktól is. A technológiai problémák azzal függenek össze, hogy egy időosztásos rendszer megfelelő sebességü és teljesitőképességü központi egységet, input/output terminálok rendszerét és megfelelő méretü központi memóriát igényel.

A lényegi problémákat azért kell megoldanunk, mert a rendszer csak igy töltheti be funkcióját, a technológiai problémákat pedig azért, mert jelenleg csak ugy képzelhető el gazdaságos megoldás, ha lehetőleg minél több felhasználó osztozik a rendszeren. Természetesen a lényegi problémák az alapvetőek és mindaddig megmaradnak, amig a kitüzött cél meg nem változik. A technológiai problémák a másodrendüek és a technikai szinvonal megváltozása a problémakör lényeges változását és átalakulását eredményezi.

Minthogy - mint az eddigi fejtegetéseinkből következik - az időosztásos rendszer alapvető célja a felhasználó és a számitógép közötti állandó kapcsolatot - interaktivitást - lehetővé tenni, a technikai szint módosulása, a gazdaságossági mutatók megváltozása, a kisszámitógépek árának további rohamos csökkenése, elvileg azt a paradox helyzetet eredményezheti, hogy egy interaktiv rendszert egyetlen felhasználó is gazdaságosan üzemeltethet, tehát nincs szükség időosztásra.

### *1.3.2. Az időosztásos rendszerek architekturája*

Egy rendszer architekturáján önállóan definiálható részeinek felépitésbeli leirását és ezen részek kapcsolatának definicióját értjük.

Egy időosztásos rendszer elsődlegesen három alapvető részre bontható:

- a számitóközpont,
- a hálózat,
- terminálok.

A továbbiakban a hálózattal és a terminálokkal nem foglalkozunk, bár egy időosztásos rendszer konkrét megvalósitásánál ezeket nagymértékben figyelembe kell venni.

A számitóközpont részei:

- operativ tárak rendszere,
- háttértárak rendszere,
- müveletvégző egységek (I/O processzorok, vagy csatornák is),
- perifériák.

Perifériákkal a továbbiakban még bizonyos vonatkozásban foglalkozunk.

Az időosztásos rendszereknél az I/O processzoroknak, illetve csatornáknak nincs alapvető jelentősége, illetve kötegelt üzemmódban is müködő rendszereknél jelentőségük azonos az ilyen rendszereknél definiált jelentőséggel.

Néhány szóban azonban utalni szeretnénk a hálózaton keresztül adatforgalmat lebonyolitó adatátviteli csatornára.

A hálózat - gazdasági okokból - egy érpárt tartalmazó vonalakból épül fel. Ez azt jelenti, hogy nincs lehetőség az adatok és a vezérlőjelek térbeli, csupán csak időbeli szétválasztására. Aszinkron hálózatoknál az időbeli szétválasztás nem történhet időzitéses alapon, csak ugy, hogy az adatforgalom üzenetekre bomlik, és az üzenetnek előre meghatározott szerkezete van. Az adatátvitelt lebonyolitó hardware-nek többé-kevésbé "ismerni" kell ezt a szerkezetet (pl. SOH - START OF HEADER, illetve EOM - END OF MESSAGE karakterek).

Másrészről azonban ma még számolni kell a terminálok "szabályozatlan sokszinüségével" is, ami igen hasznossá teszi azt a lehetőséget, hogy a csatorna az adott vonalról érkező karakterek vizsgálatát (paritásbites vagy paritásbit nélküli kódrendszer, a paritásbit páros vagy páratlan rendszerü stb.), a software által időről-időre módosithatóan végzi. Szokásos az is, hogy a csatorna bizonyos software által elhelyezett karakterek felléptét figyeli az üzenetben, és érkezésükkör I/O megszakitást produkál.

Ezen technika részleteinek tekintetében utalunk e kötet "TÁVADATFELDOLGOZÁS" cimü tanulmányára.

Magának a központi processzornak alkalmasnak kell lennie arra, hogy teljes mértékben ellenőrizze a felhasználók tevékenységét. Ehhez az szükséges, hogy néhány müveletet a felhasználó közvetlenül ne végezhessen el, csak a rezidens vezérlő software-n keresztül. Ez azt jelenti, hogy a felhasználónak nem áll a teljes utasitáskészlet rendelkezésére.

A processzornak tehát két üzemállapotának kell lennie, az egyikben minden utasitás használata megengedett - ebben az üzemmódban dolgozik a vezérlő software, mig a másik állapotban néhány utasitás használata tilos - ebben az üzemmódban dolgoznak a felhasználók.

Bizonyos okoknak a hatására a processzornak meg kell szakitania az éppen futó programot és egy adott programrészlet végrehajtását kell inicializálnia, miközben a megszakitási állapotot ugy kell "feljegyeznie", hogy a későbbiekben a megszakitott program ugy legyen tovább indítható, hogy a program "semmit ne vegyen észre". Ezt a technikát nevezzük megszakitási rendszernek (interrupt). Ezzel a megoldással a processzornak lehetősége van a felhasználó fokozottabb ellenőrzésére (pl. illegális utasitás végrehajtásának letiltása), másrészt a processzor szempontjából külső jelenségek követésére (pl. I/O müvelet vége, időintervallum letelte stb.). Ennek a hardware funkciónak minden multiprogramozható, igy az időosztásos rendszernél is alapvető jelentősége van.

Minthogy az időosztásos rendszer felhasználói tevékenységükhöz memóriakapacitást, adataik tárolására pedig tároló file-okat igényelnek, s ezt tehetik megkötés nélküli mértékben és egyidőben igen sokan, nyilvánvaló, hogy a rendszer alapvető hardware problémái a központi tárakkal és a háttértárakkal kapcsolatosak. Ezek jellemzőit ott fogjuk tárgyalni, ahol bemutatjuk, hogy a software hogyan és milyen célra használja ezeket az eszközöket.

# 2.

# Az időosztásos rendszer vezérlő softwarejének alapvető funkciói

## *2.1. A tevékenység és a virtuális processzor fogalma*

### *2.1.1. A tevékenység (process)*

Az időosztásos rendszer tárgyalásánál alapvető fontosságu a tevékenység (folyamat, process) és a virtuális processzor fogalmának tisztázása. Annál is inkább, mert különböző szakemberek, illetve különböző publikációk ezeket nem ugyanabban az értelemben használják.

A terminálokná1 ülő felhasználók tevékenykednek (programot irnak, futtatnak stb. ...), és kiszolgálásukhoz a hardware-nek és a software-nek egy konkrét rendszere szükséges. A gazdaságosság érdekében az időosztásos rendszer egyidőben több felhasználó tevékenységét teszi lehetővé, vagyis egyszerre több ilyen hardware-software rendszert tart életben. Természetesen - ugyancsak a gazdaságosság érdekében, törekedni kell arra, hogy ezen tevékenységek hardware és software elemei a lehetőséghez mérten maximálisan azonosak legyenek. A multiprogramozható kötegelt feldolgozási rendszerek lehetővé teszik, hogy a különböző tevékenységek hardware eleme nagyrészt azonos (az aritmetika, a vezérlő egység, esetleg néhány I/O csatorna stb.), de a software elemek különállóak. Az egyes tevékenységet kiszolgáló software-elemek rendszere - a program - egyedi. Ezért beszélünk ezen rendszereknél multiprogramozásról. Az időosztásos rendszerek - hogy minél több felhasználót szolgáljanak ki egyidőben - megoldják, hogy a software elemek nagyrésze is közös lehessen.

Például: ha egy multiprogramozható kötegelt feldolgozási rendszerben két tevékenység fut egyidőben, s mindkét tevékenység egy FORTRAN program forditása, a forditóprogram kétszer foglal helyet a központi tárban. Ezzel szemben a jól megoldott időosztásos rendszernél csak két FORTRAN forditó előtétprogram van, de a tulajdonképpeni forditó program mindkét tevékenység számára közös. Tehát az időosztásos rendszer több feladat egyidejü teljesitését végzi, miközben lehetővé teszi, hogy ezen tevékenységek nagyrészt azonos hardware, illetve software elemekkel rendelkezzenek, vagyis a feladatokat tulnyomórészt azonos programelemek (rutinok) oldják meg. Ezért nem lehet ebben az esetben programok szimultán végrehajtásáról beszélni, csak párhuzamosan futó tevékenységekről.

### 2.1.2. *A virtuális processzor*

Egy számitógépen futó tevékenységek hardware elemeinek összességét processzornak nevezzük. Egy közönséges, valós tárolóju multiprogramozható rendszer esetében egyetlen tevékenység sem igényelhet többet, mint ezen hardware elemek teljes rendszerét, de általában kevesebbet. Például: ha egy program memóriaigénye 64 K cimezhető elem (pl. szó vagy byte), akkor ez a program az adott számitógépen csak akkor futhat, ha annak 64 K-nál nagyobb memóriája van.

A legtöbb multiprogramozható rendszer egyébként abban az esetben, ha egy tevékenység a teljes hardware készletet vagy annak egy teljes keresztmetszetét (pl. az összes periférikus egységet, az összes memóriát) igényli, akkor elfajul monoprogramozásuvá. A virtuális processzorral rendelkező rendszerek nemcsak azt oldják meg, hogy ilyen esetben is lehetséges legyen a tevékenységek szimultán futtatása (pl. a teljes keresztmetszetet igénylő programot időnként félreteszik, s néhány tizedmásodpercig futtatják a többit, majd néhány tizedmásodpercre előveszik az elsőt stb...), de ezen tulmenően azt is, hogy olyan tevékenység is futhat a rendszerben, amely többet igényel, mint a teljes hardware. Például megoldható, hogy egy futó programnak egyszerre csak egy része legyen az operativ memóriában, s ha a program még nem memóriában lévő részét igényli, akkor a vezérlő software ezt ugy hozza be - a program bentlévő részeinek helyébe -, hogy azt a futó program "nem veszi észre". Ebben az esetben lehetséges, hogy egy 64 K memóriáju számitógépen egy 128 K méretü program - némi sebességcsökkenéssel - futhasson. Egy olyan számitógép, mely ezt a technikát lehetővé tevő hardware-software mechanizmussal van ellátva - a valóságosnál nagyobb, látszólagos - virtuális memóriával rendelkezik. Természetesen az időosztásos rendszereknél nem azért terjedt el ez a megoldás, hogy lehetőség legyen olyan nagy programok futtatására, amelyek már nem is férnének el a számitógépben, hanem azért, hogy az egyidőben memóriában levő programok minél kevesebb helyet foglaljanak el. Tehát minden program csak részlegesen helyezkedhet el a központi tárban. Hasonló gondolatmenettel juthatunk el a virtuális perifériakészlet fogalmához is. Általánosan fogalmazva: ha egy rendszerben lehetőség van arra, hogy az egyes tevékenységek által igényelt hardware elemek összessége több lehessen, mint a valóságos hardware, akkor az egyes tevékenységek hardware elemeinek összességét virtuális processzoroknak nevezzük.

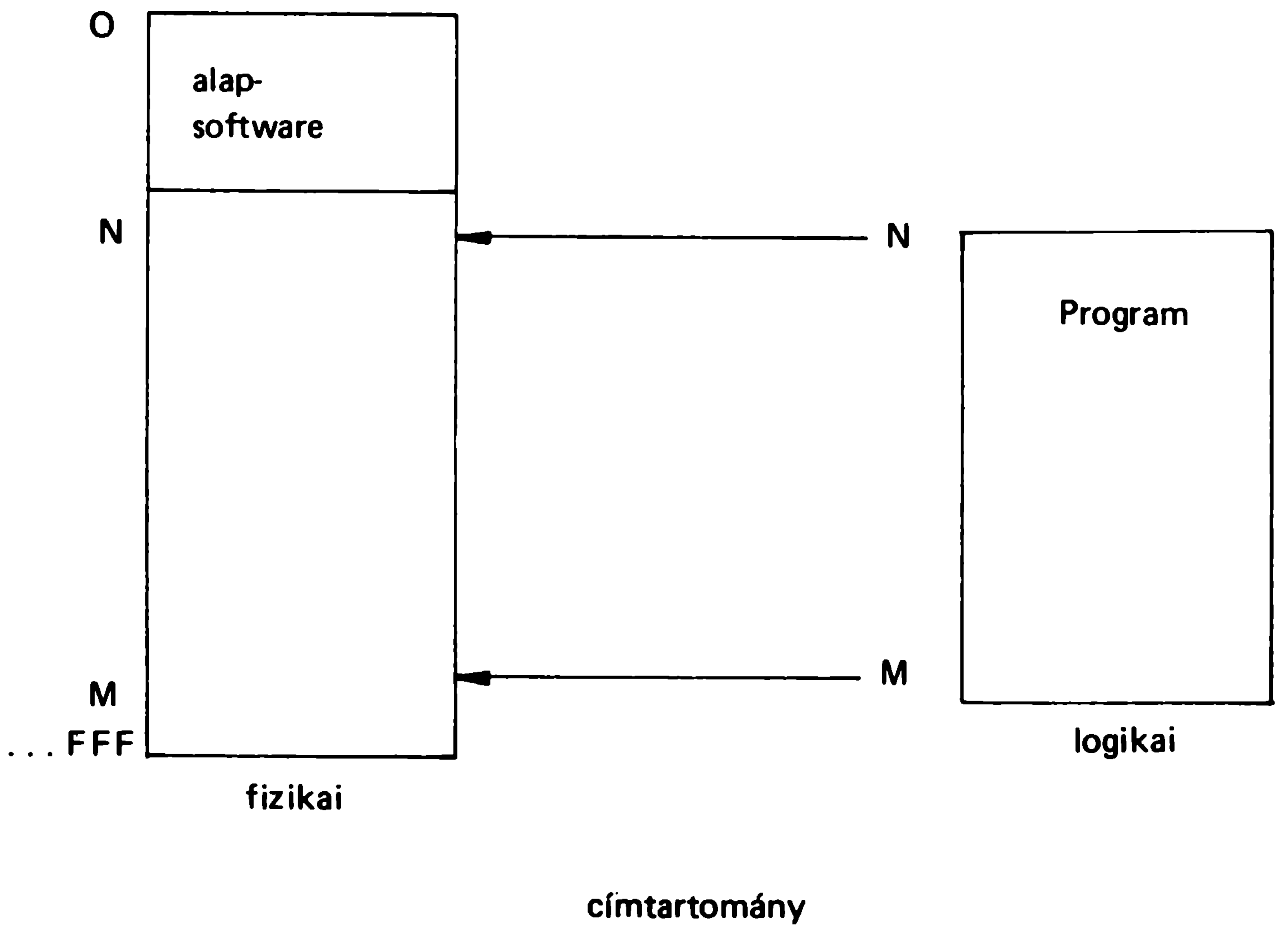
## 2.2. *Az erőforrások dinamikus allokációja*

### 2.2.1. *Bevezetés*

Az időosztásos rendszer az erőforrásoknak egy valóságos, vagy fejlettebb rendszereknél egy virtuális készletével rendelkezik. A rendszer müködése során ezt az erőforrás készletet kell pillanatról-pillanatra ujra és ujra szétosztani a rendszer által végzett tevékenységek között. A hardware erőforrások alapvető kategóriái (mint ezt az 1.3.2. fejezetben rögzitettük) az alábbiak:

az operativ és háttértárak rendszere,
a perifériakészlet,
és a müveletvégző egységek.

Ezen erőforrások pillanatról-pillanatra változó szétosztása (dinamikus allokációja) a vezérlő software rezidens része, a supervisor (vagy más gépeknél executiv, illetve monitor stb.) legfontosabb feladata. A továbbiakban az erre vonatkozó részletezés következik.



1. *ábra*
*A fizikai és logikai cimtartomány egybeesése*

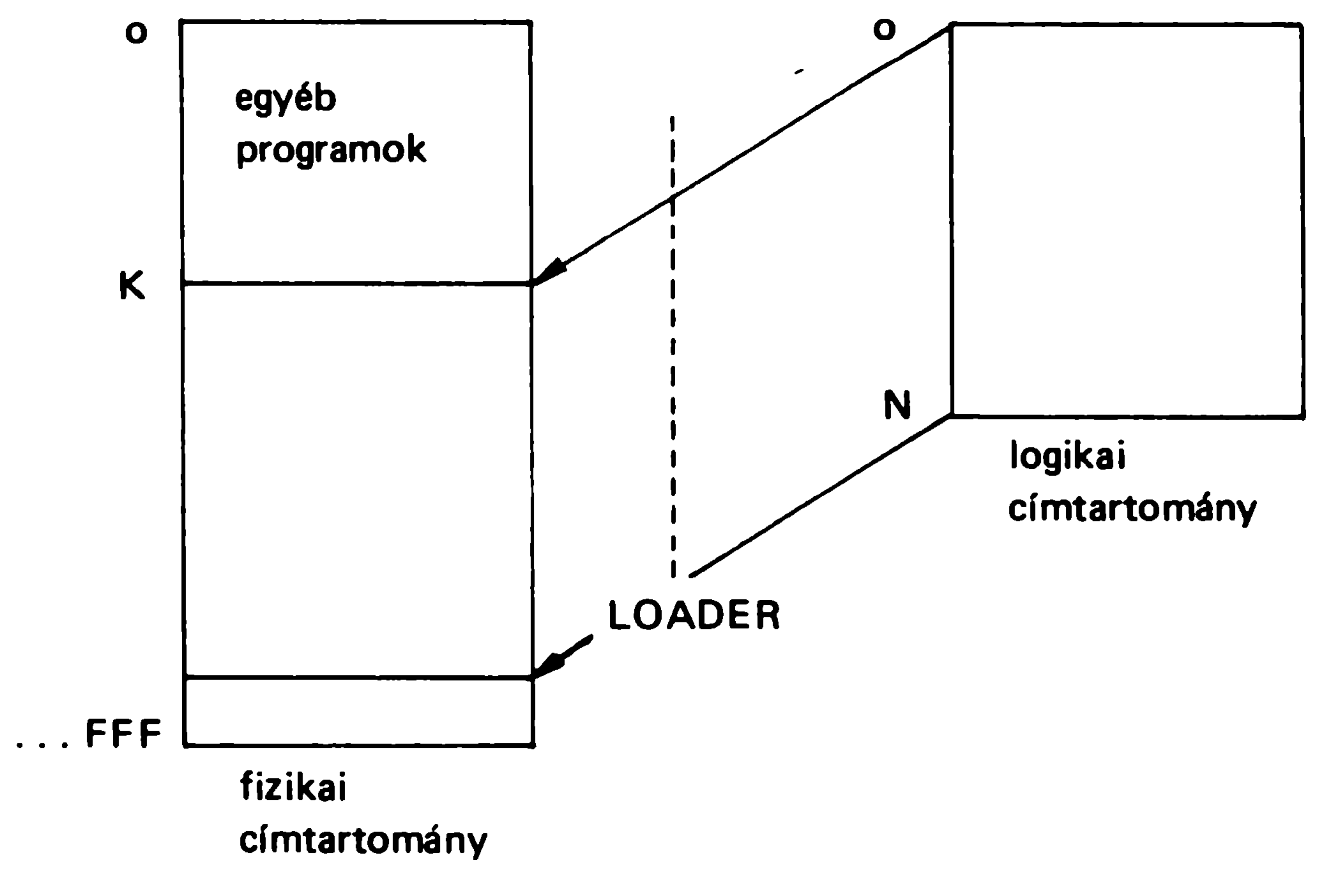
### *2.2.2. A memóriagazdálkodás problémaköre*

#### *2.2.2.1. A logikai és a fizikai cimtartomány összefüggése*

A számitógép operativ memóriája cimezhető egységekből áll. A lehetséges cimek 0-tól a max. memória cimig folyamatosak. A számitógépen futtatandó programban ugyancsak minden utasitásnak, adatmezőnek cime van, a lehetséges cimek a program legkisebb cimétől a legmagasabb ciméig tartó intervallumba esnek. Az előbbi a központi memória fizikai cimtartománya, az utóbbi pedig a program logikai vagy cimzési tartománya. /Irod. 2. /

A legegyszerübb, monoprogramozási rendszereknél a kettő azonosan egyenlő lehet. Amennyiben egy ilyen számitógép alapsoftware-t is használ, a programnak ezt a cimtartományt szabadon kell hagynia (1. ábra).

Multiprogramozható gépeknél ez a megoldás nyilvánvalóan nem használható, hiszen a futtatáskor derül csak ki, hogy milyen programok futnak egymás mellett. A fizikai és a logikai cimtartomány tehát szétválik. A program futásakor azonban egy leképező mechanizmusnak létre kell hoznia a logikai cimtartomány megfeleltetését



*2. ábra*
*A logikai cimtartományt a LOADER leképezi a fizikai cimtartományra*

a fizikai cimtartomány egy részével. A megfeleltetés történhet a program memóriába történő betöltésekor. Ebben az esetben a futáskor már minden utasitásban, illetve minden olyan konstanstároló mezőben, melynek tartalma a program helyétől függ (cimkonstansok), az eredeti programrelativ cim ki van cserélve egy konkrét fizikai cimmel. A cserét a betöltést végző software elem (static loader) végzi el a forditóprogramtól kapott információk alapján (2. ábra).

Ez a statikus leképezés azonban nem teszi lehetővé a programok áthelyezhetőségét. Ennek tárgyalása a következő pont feladata.

### *2.2.2.2. A logikai és a fizikai cimtartomány dinamikus megfeleltetése*

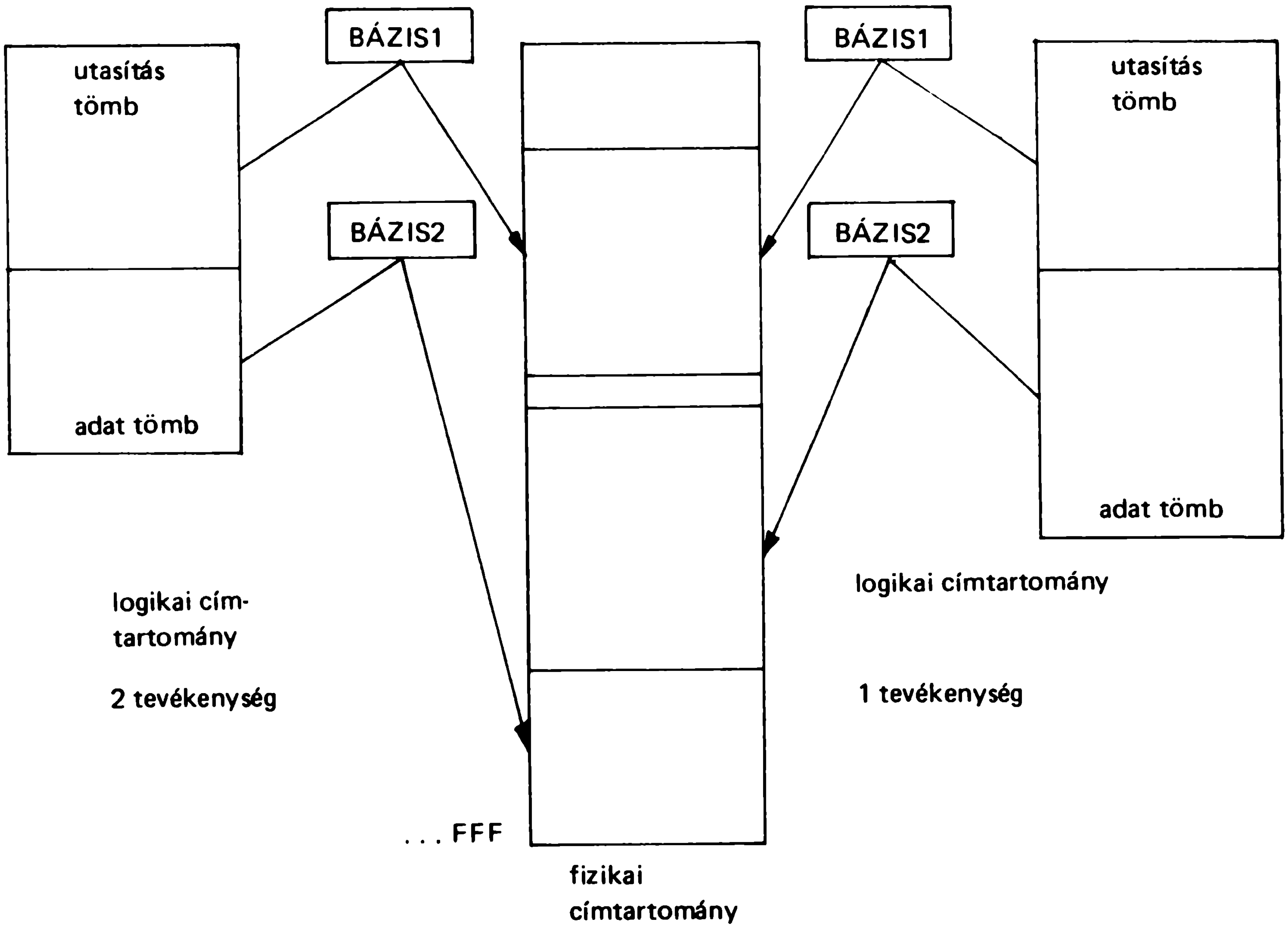
Az előzőekben leirt leképezési mód az utasitások és bizonyos adatmezők loader általi átirását jelenti. Az ilyen programokat a memóriában áthelyezni nem, vagy csak bizonyos megkötések árán lehet. A programok memóriabeli áthelyezéséről azonban az időosztásos rendszer nem mondhat le, hiszen szükség lehet a különböző helyeken a fizikai cimtartományban felszabadult blokkok összetolására, vagy - mint ezt később látjuk majd - egy program eltávolitására, majd későbbi visszatöltésére. Nem remélhető, hogy a visszatöltés idején az eredeti hely szabad. Más helyre történő visszatöltés azonban a program áthelyezésével egyenértékü. Ezen probléma megoldására kidolgozott megoldások közös jellemzője, hogy a program memóriába történő betöltésekor nem változik, cimhivatkozásai az eredeti logikai cimhivatkozások maradnak, tehát a program nem tudja, hogy a memóriába hová lett betöltve. Természetesen léteznie kell egy hardware mechanizmusnak, amely az utasitás végrehajtásakor meghatározza a logikai cimhez tartozó fizikai cimet.

Az egyik lehetséges megoldás a bázisregiszterek használata. Ebben az esetben minden programhoz tartozik egy regiszter, melynek tartalma a logikai cimhez hozzáadódik, és igy alakul ki a fizikai cim. A program memóriabeli áthelyezésekor csak a bázisregiszter tartalmát kell a supervisor-nak átirni és a program futhat tovább.

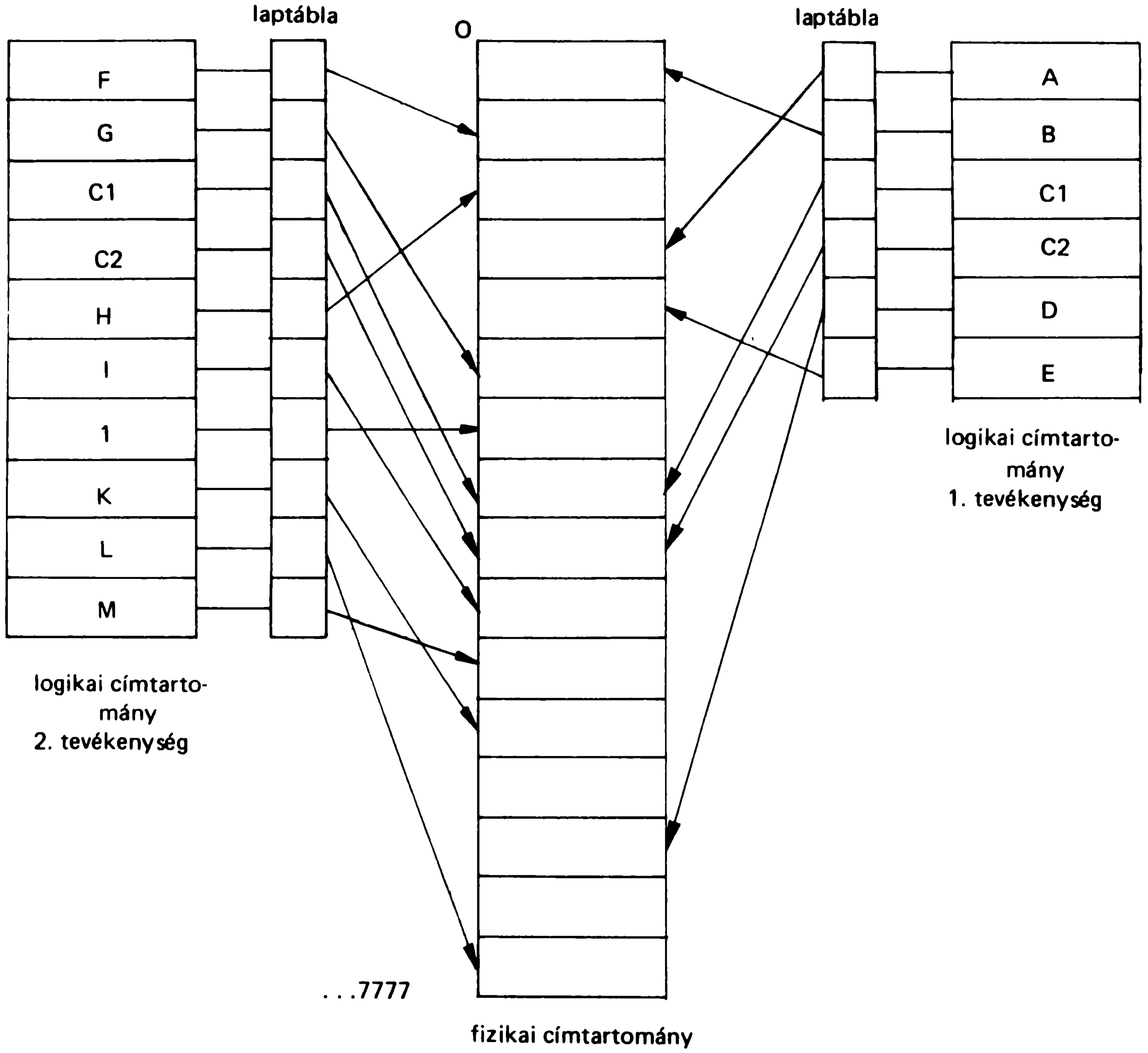
Ez a technika tovább finomitható több bázisregiszter használatával. Az időosztásos rendszerek fejlődése szempontjából különösen érdekes a következő változat: biztositsuk, hogy a programban az utasitások tömbje és az adatok tömbje külön váljon. Rendeljünk minden programhoz két bázisregisztert: egyet az utasitástömb, egyet pedig az adattömb leképzéséhez (3. ábra). Ezzel a megoldással az előbbiekben leirt lehetőségeken tulmenőleg lehetőség van arra, hogy egy software elem szimultán üzemben két adattömbön munkálkodjon. Ha az adott software elem az egyik tevékenységhez tartozik, akkor utasitás bázisregiszterébe a saját bázisát, adatbázisregiszterébe pedig a megfelelő tevékenység adatmezejének kezdő cimét kell betölteni. Természetesen ilyen egyszerre két (vagy még több) tevékenységet végző software elem felépitésénél bizonyos szabályokat kell figyelembe venni (ld. 2.3.1. pont). A bázisregiszterek bevezetésével tehát nemcsak a programáthelyezést, hanem a közös software elemek használatának lehetőségét (Software Sharing) is feltártuk.

Ugyanezen előnyökhöz juttat egy rendszert a laptechnika használata is. Ez a hardware mechanizmus is az utasitás konkrét végrehajtása idején határozza meg a logikai cimnek megfelelő fizikai cimet. Továbbiakban leirjuk a mechanizmus müködését:

Képzeljük el, hogy a számitógép memóriájának fizikai cimtartománya egyenlő hosszuságu (mondjuk 2 K szó nagyságu) darabokra van osztva. Nevezzük ezeket a darabokat fizikai lapoknak. Osszuk fel a rendszer által futtatott programok logikai cimtartományait ugyanilyen hosszuságu darabokra - a logikai lapokra. Ha létezik a számitógépen egy olyan mechanizmus, mely a futás idején az egyes tevékenységek (process) logikai cimtartományát alkotó logikai lapokat hozzárendeli a fizikai cimtartomány egy-egy lapjához, akkor a feladat meg van oldva. Ez a hardware mechanizmus a laptáblán keresztüli cimmeghatározás.


utasítás tömb
adat tömb
BÁZIS1
BÁZIS2
logikai cím-tartomány
2 tevékenység
...FFF
fizikai címtartomány
BÁZIS1
BÁZIS2
utasítás tömb
adat tömb
logikai címtartomány
1 tevékenység

*3. ábra*
*A logikai cimtartomány leképzése a fizikai cimtartományra bázisregeszterek segitségével*


laptábla
O
laptábla
F
G
C1
C2
H
I
1
K
L
M
A
B
C1
C2
D
E
logikai címtartomány 1. tevékenység
logikai címtartomány 2. tevékenység
...7777
fizikai címtartomány

*4. ábra*
*A logikai cimtartomány leképzése a fizikai cimtartományra laptábla segitségével*

Tételezzük fel, hogy minden tevékenységhez tartozik egy táblázat, melynek annyi eleme van, ahány logikai lapból áll az illető tevékenység. A tábla egyes elemei az egyes logikai lapokhoz rendelt fizikai lapok sorszámát tartalmazzák. Amennyiben a számitógép egy konkrét utasitást végre akar hajtani, akkor az utasitásban szereplő logikai cim utolsó bitjeit (2 K nagyságu lapok esetén utolsó 10 bitet) levágva, meghatározza a logikai lap sorszámát, megkeresvén a laptáblában az ehhez tartozó fizikai lap számát, majd az igy kapott fizikai cimbiteket kiegészitve az utolsó (pl. utolsó 10) bitek által reprezentált lapon belüli cimmel, megkaptuk a konkrét fizikai cimet (4. ábra). A konkrét hardware megvalósitással e sorozat későbbi tanulmányai részletesen foglalkoznak majd. A mi szempontunkból a döntő kérdés ezzel kapcsolatban az, hogy milyen lehetőséget ad ez a technika az időosztásos rendszer megvalósitására.

Nyilvánvaló, hogy - hasonlóan a bázisregiszteres megoldáshoz - a program itt sem tud arról, hogy hol helyezkedik el a memóriában, tehát akár laponként is áthelyezhető bárhova, mindössze csak a laptáblában kell az áthelyezést adminisztrálni. Elképzelhető azonban, hogy egy tevékenységnek nincs az összes logikai lapja a memóriában, ez nyilvánvalóan azt kell jelentse, hogy a laptábla bizonyos elemei nem tartalmaznak fizikai lapszámot. Ha a hardware a konkrét cimkidolgozás során ezt az esetet felismeri, s megszakitva a tevékenység futását, a rezidens vezérlő software-t értesiti, akkor annak lehetősége van az igényelt lapot valamilyen másodlagos tárolóeszközről a memóriába másolni és ezt a laptáblában feljegyezve, az eredeti program továbbfutását ugy lehetővé tenni, hogy az semmit sem vett észre. Ezzel a megoldással lehetségessé vált egy látszólagos fizikai memória használata (ld. 2.1.2. pont).

A virtuális tár lehetősége igen vonzó az időosztásos rendszerek tervezői számára, ezért nem véletlen, hogy az ilyen üzemmódu számitógépek túlnyomó többsége ilyen felépitésü, sőt manapság egyre több kötegelt üzemmódu rendszer is igyekszik előnyeit élvezni (pl. IBM 370 VS1 ill. VS2). Egy másik alapvetően fontos funkció a software elemek közös használata (Software Sharing). Lássuk, hogyan lehetséges ez a laptechnika használata mellett! Tételezzük fel, hogy két tevékenység fut egyidőben a rendszerben, s mindkettő egy-egy forrásprogramot fordit. Mindkét tevékenység használja tehát a forditóprogramot (ld. 4. ábra). Jelöljük az ábrán a forditóprogramot C1 és C2-vel, (feltételezvén, hogy mérete 2 lapnyi).

Bár ez a program mindkét tevékenységnek része, csak egyszer kell a memóriában elhelyezkednie, mert mindkét tevékenység laptáblája ugyanazon fizikai lapokat rendeli a C1 és C2 jelü logikai lapokhoz. Rá kell azonban mutatni egy nehézségre. A közös forditóprogram a forditandó forrásprogramot logikai cimtartománya szerinti helyen keresi. Ez a cimtartomány a forditóprogram betöltésekor rögzitődött. A két tevékenységnek logikai cimtartományában ugyanott elhelyezkedő és ugyanolyan méretü adattömböt kell tartalmaznia. Ha a két tevékenység több közös software elemet is használ, akkor ezek a megkötések még bonyolultabbak, s végül odavezetnek, hogy fel kell épiteni egy elvi logikai cimtartomány rendszert, melyben minden közösen használható software elemnek helye van, s minden közösen használható adatterület mérete rögzitett. Ha valamelyik tevékenység bizonyos közösen használható software elemeket nem igényel, akkor ezek logikai lapjait sohasem érinti, ebben az esetben a vezérlő software ezeket a lapokat nem felelteti meg a fizikai tár egyik lapjának sem, és igy a virtuális memória feloldja a logikai cimtartomány nem használt lyukjait. További sulyos probléma azonban a logikai cimtartomány - és ebből következőleg a cimbitek számának - tulzott megnövekedése. Ezt a helyzetet a szegmentálás (ld. 2.3.3.) oldja meg.

Lehetséges egy másik megoldás is, melynél a közösen használt software elem, nem közvetlenül része az egyes tevékenységeknek, hanem önálló laptáblával rendelkezik, és pl. a supervisor segitségével hivható. A megoldandó feladat ebben az esetben annak biztositása, hogy az adott software elem egyszer az egyik, másszor a másik adattömbön dolgozzon. Nyilvánvalóan ez megoldható, ha az adattömb külön logikai lapon, vagy lapokon van, és a laptáblában egyszer az egyik tevékenység adattömbjének fizikai cime van, másszor a másiké. Lehetséges további finomitás: a közösen használt software elem bizonyos logikai lapjait a saját laptábláján, mig más logikai lapjait a hivó tevékenység laptábláján keresztül képezi le a fizikai cimtartományra. A két bázisregiszteres megoldás ugyanugy lehetővé teszi közös software elemek használatát, mint a laptechnika. Ez utóbbi - a virtuális tár bevezetésével további előnyökkel kecsegtet, bár a virtuális tár adminisztrálása bonyolult, és az input/output bonyolitásánál sok megkötéshez vezet. Talán ez is az oka, hogy manapság az egyik legsikeresebb időosztásos tevékenységet lehetővé tevő multiprogramozható kötegelt feldolgozási rendszer a UNIVAC cég 1100-as gépcsaládján implementált EXEC 8 rendszer, amelyik a 3. ábrán bemutatott két bázisregiszteres technikával dolgozik.

### *2.2.2.3. Hierarchikus tárak*

Már a kötegelt feldolgozást végző multiprogramozható rendszerek architekturájának is alapvető problémája volt a megfelelő méretü operativ memória biztositása. Az időosztásos rendszereknél ez a probléma még égetőbb. Az operativ gyorsmemória a számitógép legköltségesebb erőforrása, s az időosztásos rendszerek éppen ebben a tekintetben a legigényesebbek. Bár az előző paragrafusban ismertettük azokat a módszereket, amelyekkel az operativ memória kiaknázása nagymértékben fokozható, az éppen nem futó programok kimentésére, visszamentésére, vagy a nem használt logikai lapok őrzésére stb., gyors tárolási, illetve visszatöltési lehetőségre van szükség. E feladat megoldására több lehetőség is kinálkozik, melyek alternative, vagy egymással kombinálva választhatók.

A ferritgyürüs memória gyártási költsége a memória elérési idejének hosszától forditott módon függ. Tehát a gyors memória drágább, a lassu jóval olcsóbb. Jó teljesitmény/ár viszony érhető el a kétféle memória kombinálásával. Például: a CDC 7600-as számitógép gyorsmemóriája 65 K-szó nagyságu, ciklusideje 27,5 ns; mig a nagy memória mérete 512 K-szó, ciklusideje pedig 176 ns. Ez a számitógép a memóriagazdálkodás, programáthelyezés stb. problémákat csupán memória hierarchia felhasználásával oldja meg. A gyors memóriában (SCM = Small Core Memory) csupán egyetlen felhasználói program foglal helyet, ezen kivül még itt van természetesen a vezérlő software rezidens része, és az I/O pufferek. A központi vezérlőegység mindig az SCM-ben lévő program utasitásait hajtja végre. Ha a futó programhoz rendelt időintervallum letelt, vagy a program folytatását rövid időre fel kell függeszteni, akkor kicserélődik a lassu memóriában (LCM = Large Core Memory) lévő programmal.

Az LCM-ben egyidőben két program tartózkodhat a rendszer különböző paramétertáblázatain és rutinkönyvtárán kivül. Ha valamelyik program futtatását hosszabb időre fel kell függeszteni, akkor az LCM-ből is kikerül. Az itt vázolt megoldással tulajdonképpen minden relokálási probléma megkerülhető, sőt közös software elemek használatára sincs szükség. Az összes eddigi multiprogramozási problémát visszavezettük az alapkérdéshez, az input/output időigényének feloldásához, ezt pedig 8, csupán input/output feladatokat végző számitógép beállitásával oldották meg. Ez a megoldás a hierarchikus memóriahasználat érdekes, szélsőséges példája, megvalósitása azonban igen költséges. Kevésbé szélsőséges módon több közismert időosztásos rendszer támaszkodik hierarchikus felépitésü memóriára (pl. UNIVAC 1110).

### *2.2.2.4. Információcsere (SWAPPING)*

A minél több program egyidejü futtatására törekvő rendszereknél a memóriában lévő és ideiglenesen tovább nem futtatható (blokkolt) programokat mágneses háttértárra mentik ki, illetve hozzák vissza. Ez a mágneses háttértár régebben általában mágnesdob volt, ma már jobbára mágneslemez. Természetesen csak random tárolást lehetővé tevő eszközök jöhetnek szóba, s ezek közül csak azok a tipusok, amelyek a kiválasztott blokk gyors kikeresését minél nagyobb sebességgel képesek elvégezni. (A szokásos adatok: átlagos elérési idő 5-35 msec, átviteli sebesség 300 000 - 1000 000 karakter/sec.)

Az operativ memória és a mágnes háttértár közötti programcserét szokták swapping-nek nevezni. Meg kell azonban jegyezni, hogy ugyanerre a funkcióra elterjedt a "roll-in roll-out" kifejezés is. A különböző cégek a különböző kifejezéseket különböző értelemben használják. Az előző paragrafusban emlitett CDC-7600-as rendszernél az operativ memória (SCM) és a másodlagos ferritgyürüs memória (LCM) közötti programcserét hivják swapping-nek, mig a másodlagos ferrit és a mágnesdob közötti programcserének a neve roll-in roll-out.

A UNIVAC cég az elsődleges memória és a mágneslemezes háttértár közötti programcserét nevezi "roll-in roll-out"-nak. (Az elterjedt magyar szóhasználat: swapping-programrészek, adatrészek cseréje, mig a roll-in roll-out - komplett programok mozgatása.)

Swapping esetén, tehát teljes programot vagy programrészeket (laptechnika) cserélünk. Ezzel kapcsolatban a következő lényeges kérdésekre kell választ adnunk.

- Mikor hozzunk be egy uj programot?
- Mit tegyünk, mialatt a swapping folyik?
- Mely program terhére hozzuk be az uj programot?

A mai modern, időosztásos rendszerek egyidőben több programot tartanak az operativ memóriában. Ezek a programok háromféle állapotban lehetnek:

éppen futó (természetesen egyetlen ilyen lehet),
futásra készen álló,
eseményre váró (blokkolt).

A rendszerek igyekeznek a gondjukra bizott tevékenységeket szimultán futtatni, ezért a futásra kész programtól is elveszik néha "az időt", és egy másik program kerül éppen futó állapotba, természetesen az előbbi, és esetleg még igen sok program futásra készen álló. Ezzel szemben a programoknak időnként bizonyos események lezajlását meg kell várniuk, a leggyakoribb ilyen esemény egy input/output müvelet lezajlása. Az ilyen programok bizonyos ideig továbbfuttatásra alkalmatlanok - blokkoltak.

A különböző rendszerek megegyeznek abban, hogy csak az operativ tárban lévő egyik program blokkolásakor kezdeményeznek swapping-ot.

Az egyszerübb rendszerek, különösen azok, amelyek az operativ tárban csak egy programot tárolnak, a swapping alatt nem tudnak hasznos munkát végezni, meg kell várniuk, amig a futtatandó uj program be nem töltődött. Ezen rendszerek némelyike az időveszteséget azzal próbálta csökkenteni, hogy ha a behozott program kisebb, mint az előzőleg benntartott, akkor az előző program át nem fedett részét nyilvántartotta, és ha a következő swapping-nél történetesen az előző programot kellett visszahozni, akkor csak az átfedett (másik programmal átirt) területet frissitette fel. Természetesen a rendszer hatékonysága megkivánta, hogy a swapping alatt a rendszer hasznos munkát végezzen. A programok relokálhatóságának megoldása lehetővé tette, hogy a memóriában egyidőben több program tartózkodjék, igy lehetőség van arra, hogy a programcsere alatt, egy memóriában lévő másik program dolgozzon.

Igen ám, de nem mindegy, hogy melyik program dolgozik, melyik jön be és melyik megy ki. E kérdések átgondolt megválaszolásán mulik a rendszer hatékonysága. Egy egyszerü algoritmus a következő: nézd végig a futásra készen álló programok listáját, ha a legmagasabb prioritásu ezek közül a memóriában van, inditsd el, ne csinálj programcserét; ha nincs a memóriában inditsd el behozatalát és add át vezérlést a *memóriában lévő* legmagasabb prioritásunak; ha a swapping ideje alatt ez a program is blokkolódik, akkor add át a vezérlést a következőnek. /Irod. 4./

Ha a memóriában lévő programok input/output igényesek, gyakran előfordulhat, hogy a programcsere ideje alatt minden addig futásra készen álló program blokkolódik és a rendszer várakozni kényszerül. Ésszerűnek tünt az algoritmus továbbfejlesztése olyan irányban, hogy a swapping alatt számitásigényes programot futtassunk, mert az előreláthatólag nem fog blokkolódni. A rendszernek tehát kétféle nyilvántartást kell vezetnie, egyet a futásra készen álló input/output igényes programokról (I/O bound), másikat a hasonló számitásigényes (compute bound) tevékenységekről. Ezután a választott algoritmusnak biztositania kell azt, hogy a memóriában mindig legyen egy (vagy több) számitásigényes program, és a programcsere ideje alatt ezeket kell futtatni, még akkor is, ha relative kicsi a prioritásuk. /Irod. 5./

Nyitott maradt a kérdés, hogy honnan tudja a vezérlő rendszer, hogy melyik tevékenység számitásigényes és melyik nem. Amint ezt már futólag emlitettük, a szimultán futtathatóság miatt az egyes programok csak egy adott "időszelet" leteltéig futhatnak, ezután más programok kerülnek - hasonló körülmények mellett - sorra. Ha a program input/outputra kényszerül várni, akkor még az időszelet letelte előtt visszaadja a rezidens vezérlősoftware-nek a központi egységet. Nomármost az utóbb leirt rendszer az olyan programot, amely általában kimeriti időszeletét - a számitásigényes kategóriába sorolja, mig a másikat input/output igényesnek nevezi.

Hasonló módon fejlődött a válasz, a harmadiknak felvetett kérdésre is, tehát arra, hogy melyik program terhére hozzuk be az uj programot. Eleinte a legrégebben memóriában lévő program által elfoglalt helyet szabaditották fel, később a memóriában levő legalacsonyabb prioritásu input/output igényes programot törölték. A memóriában lévő számitásigényes program addig marad, amig nem blokkolódik, akkor azonnal ki kell cserélni egy másik számitásigényes programmal.

Amennyiben a hardware lehetővé teszi a laptechnika, és ezáltal a virtuális tár használatát, akkor további kérdésekkel is szembe kell nézni. Ezek közül a legfontosabbak:

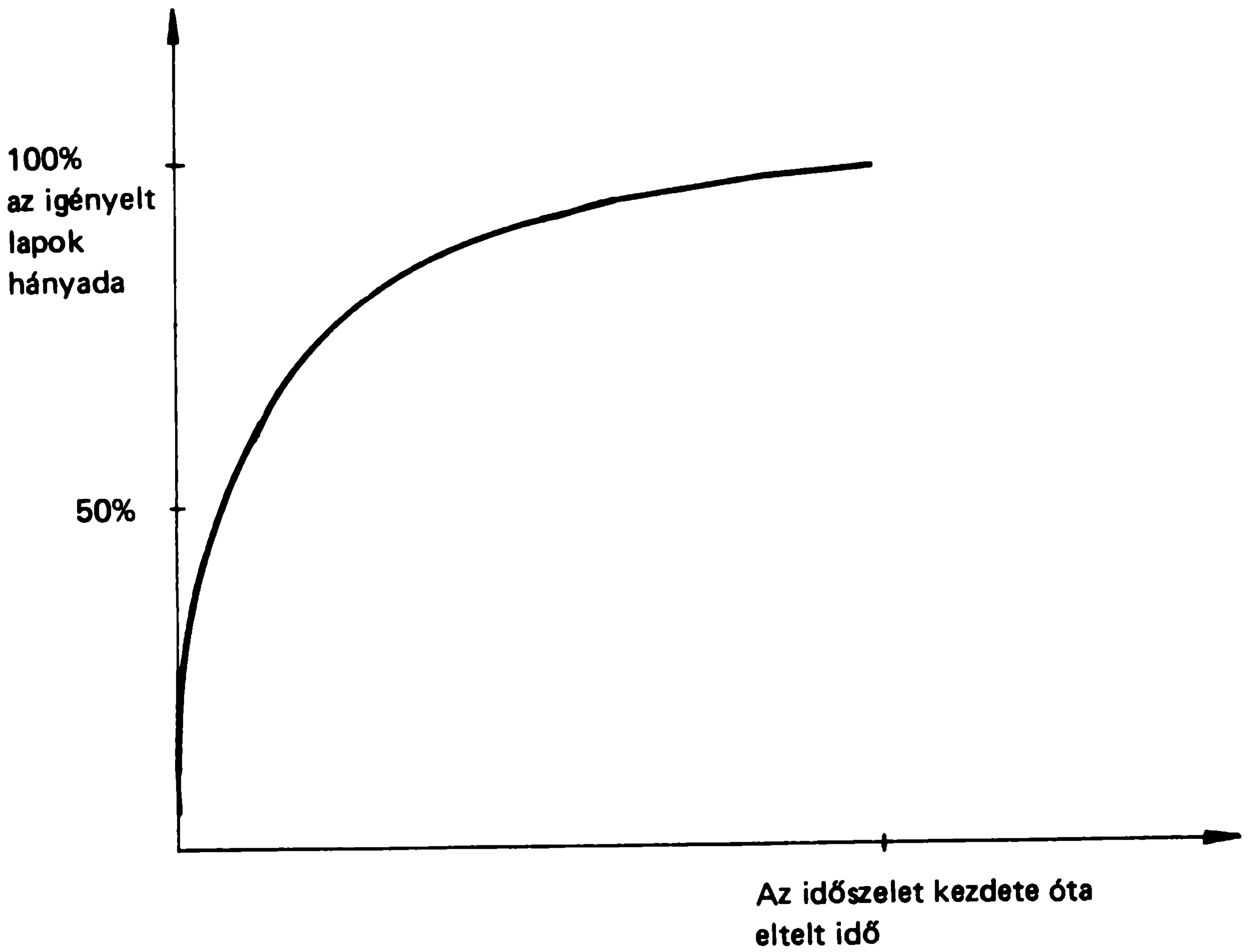
- hány lapját hozzuk be egy ujonnan betöltendő tevékenységnek?
- mely lapokat vigyük ki a háttértárolóra, hogy a szükséges memória területet felszabaditsuk?

Ugy tünik, hogy az előbbi kérdésre a kézenfekvő válasz, hogy egy uj programnak csak egy lapját kell behozni, a többit majd akkor, és olyan sorrendben, amikor és ahogy kéri (v.ö. 2.2.2.2. pont). Hogy ez a stratégia célravezető-e, az három tényezőtől függ:

a program laphasználati karakterisztikájától;
a lapváltás adminisztrálásához szükséges software időtől;
a lapváltás elvégzéséhez szükséges hardware időtől.

Az első problémával kapcsolatban a kisérletek meglehetősen lehangolók /Irod. 6./ A bemutatott diagramm tanusága szerint (5. ábra) egy átlagos - a laptechnika adottságait figyelembe nem vevő - program időintervallumának első harmadában lapjainak 75 %-át igényli. Az igény szerinti lapbehozás algoritmusa tehát nagy lapforgalmat fog okozni; hogy ez gazdaságos-e, és hogy az időosztásos rendszer nem válik-e lapváltással foglalkozó rendszerré, az döntően függ a konkrét géptől és az alkalmazott software-től.

Másodiknak vetettük fel azt a kérdést, hogy mely lapokat célszerü az ujakkal lecserélni. Természetesen másképpen kell eljárni azokkal a lapokkal, amelyek közösen használt software elemekhez tartoznak, és másképpen az egyéb lapokkal. Az egyéb lapok cseréjére a legegyszerübben implementálható algoritmus: amelyik lap legelőször jött be (legrégebben van tehát a memóriában), az lépjen ki elsőnek (first in, first out = FIFO), látható azonban, hogyha a legrégibb lapok még intenziv használatban vannak, akkor ez a technika sulyos időveszteséghez vezet.



*5. ábra*
*Egy átlagos laphasználati karakterisztika*

Finomabb módszer annak a lapnak a lecserélése először, amelyiket a rendszer legrégebben *nem használta* (least recently used = = LRU). Ennek az algoritmusnak a megvalósitása azonban igen nehéz, sőt hardware segitség nélkül lehetetlen. Egy megoldás pl. a következő: minden lapnak van egy korjelző regisztere (aging register). Ha a rendszer használ egy lapot, akkor a regiszter legfelső bitje 1-be áll. A hardware gondoskodik arról, hogy az összes lap korjelző regisztere adott időintervallumonként egy bittel jobbra legyen léptetve. Ha egy laphoz tartozó regiszter tartalma üres, ez azt jelenti, hogy a rendszer által számontartott időn belül ez a lap használaton kivül volt, tehát cserére javasolható.

### *2.2.3. A perifériagazdálkodás problémaköre*

Bár az input/ouput lebonyolitása az időosztásos rendszereknek bonyolult problémája, a perifériagazdálkodás általában egyszerü. Az előbbi állitás érzékeltetésére talán elég annyi, hogy ha egy tevékenység I/O müveletre vár, akkor blokkolva van, és általában kikerül a memóriából.

A perifériák azonban csak a központi memóriával tudnak kommunikálni. Tehát a tevékenységhez tartozó puffereket, amelybe, illetve amelyből a perifériák dolgoznak, nem szabad megszüntetni, háttértárba kimásolni, egészen addig, mig az I/O müvelet be nem fejeződött.

A perifériagazdálkodás ezzel szemben azért egyszerü az időosztásos rendszernél, mert ez nem olyan feladat, amelyet "láthatatlanul" kell megoldani. A felhasználó file-okat létesithet, kezelhet stb. és itt nem szabad észrevennie, hogy más is használja a számitógépet, de ez nem perifériakezelés.

Néhány időosztásos rendszer különleges szolgáltatásként lehetővé teszi, hogy a felhasználó kötegelt feldolgozásnál megszokott perifériákat használjon (pl. sornyomtató, kártyaolvasó, mágnesszalag stb.), de az ilyen perifériákat a felhasználónak igényelnie kell és a rendszer megtagadhatja a felhasználó ilyenértelmü kiszolgálását. Például: az XDS-940-es rendszernél, ha a felhasználó sornyomtatót kér, és nincs szabad sornyomtató, akkor ezt egy "printer in use" (sornyomtató foglalt) üzenet jelzi számára. A felhasználónak addig kell ismételnie kérését, mig meg nem kapja a kért eszközt. Más rendszerek tudomásul veszik a kérést, tárolják, és amikor ki tudják elégiteni, jelzik a felhasználónak. A MULTICS rendszernél pedig a felhasználó - bejelentkezése után - érdeklődhet, hogy a kért periféria szabad-e, s ha nem, megrendelheti egy adott időpontra. Ha a kért időpontra nincs még előjegyzés, akkor a rendszer elfogadja a "rendelést". Ezután, ha a felhasználó a megadott időpontban ujra bejelentkezik, megkapja a kért perifériát.

Másik lehetséges megoldás, hogy a rendszer minden "egy időben csak egy felhasználó által használható" perifériát mágneslemezen vagy mágnesdobon szimulálja. Ezt a technikát "spooling"-nak szokás nevezni. Ezzel a módszerrel egy időosztásos rendszer még az ilyen tipusu perifériák szétosztását is a felhasználó számára "átlátszóan" tudja elvégezni. Természetesen a rendszer részeként müködő un. Symbiont programokkal végül a mágneslemez vagy mágnesdob tartalmát fel kell dolgoztatni, és a sornyomtatón a táblát vagy a lyukkártya lyukasztón a kártyacsomagot, sőt esetleg a mágnesszalagos file-t el kell készittetni. A rendszer az erre vonatkozó adatokról tájékoztatja a központi konzolnál ülő vezetőoperátort. Ezzel a módszerrel dolgozik pl. a PDP 10-es gépen megvalósitott időosztásos rendszer. Ez az utóbbi módszer egyébként - az általában ugyis szük keresztmetszetet jelentő - lassu perifériák kiaknázását nagymértékben elősegiti, ezért a közönséges multiprogramozásu kötegelt feldolgozási rendszereknél is igen elterjedt.

### 2.2.4. *Gazdálkodás a központi egység idejével*

Bár valamelyik tevékenység programutasitásainak végrehajtása közben, ezzel átlapolva, a rendszerben sok különböző müvelet - elsősorban input/output transzfer - folyik, minthogy az esetek tulnyomó többségében csak egy központi egység van, egyidőben csak egy program futhat. Az eddig ismertetett eljárások segitségével azt is biztositani tudjuk, hogy a központi egység minél kevesebb ideig maradjon tevékenység nélkül, bármely program blokkolódásakor át tudunk kapcsolni egy másikra, és ugy cseréljük közben a programokat, hogy a bentlevők blokkolódása lehetőleg sohase legyen általános. Ez azonban még mind nem elég. Már a multiprogramozásu kötegelt feldolgozási rendszereknél kiderült, hogy elsősorban a perifériák intenziv kihasználása érdekében a nagy input/output tevékenységü programokat célszerü számitásigényes programokkal szimultán futtatni.

Azonban az utóbbiak "önszántukból" huzamos ideig sem engedik át a központi egységet más programoknak, ezért célszerü volt egy ujabb programmegszakitási okot a számitógépbe beépiteni, hogy egy adott idő eltelt után a vezérlősoftware-nek lehetősége legyen a helyzetet áttekinteni.

A számitógép egy bináris órát kapott, mely általában millisecundum-onként számlál egyet, s ha a megadott idő letelt, megszakitja az éppen futó program végrehajtását.

A vezérlősoftware mikor egy tevékenységet elindit, a prioritási osztálynak, végrehajtási kategóriának (számitásigényes - I/O igényes) stb. megfelelően egy időszeletet, - egy adott időintervallumot - oszt ki. Ha a program I/O végrehajtása miatt fordul a vezérlő software-hez, akkor ezért; ha pedig a hozzá rendelt időintervallum letelt, akkor azért kerül a vezérlés a supervisorhoz, amelynek igy lehetősége van egy másik tevékenységet is elinditani. Tulajdonképpen ez a time-sharing rendszerek nevének forrása.

## *2.3. A programok helyigényének minimalizálása*

### *2.3.1. Reentrant software*

A tevékenység fogalmának tisztázásakor - (2.1.1. pont) rámutattunk arra, hogy egy időosztásos rendszernél egy-egy software elem (program) több különböző tevékenységhez tartozhat. Előfordulhat, hogy a rendszerben hosszabb-rövidebb ideig ugyanazon program fut - de hol az egyik, hol a másik tevékenységhez tartozóan. (Ezért nem célszerü, programok szimultán futtatásáról - multiprogramozásról - beszélni.) Megmutattuk, hogy az időosztásos rendszer tervezői törekednek arra, hogy lehetővé tegyék több tevékenységnek ugyanazon programelem használatát. A logikai és a fizikai cimtartomány egymáshozrendelésének tárgyalásakor (2.2.2.2. pont) áttekintettük a szóbajöhető hardware megoldásokat. Láttuk, hogy a két bázisregiszteres megoldás ezt éppugy lehetővé teszi, mint a laptechnika.

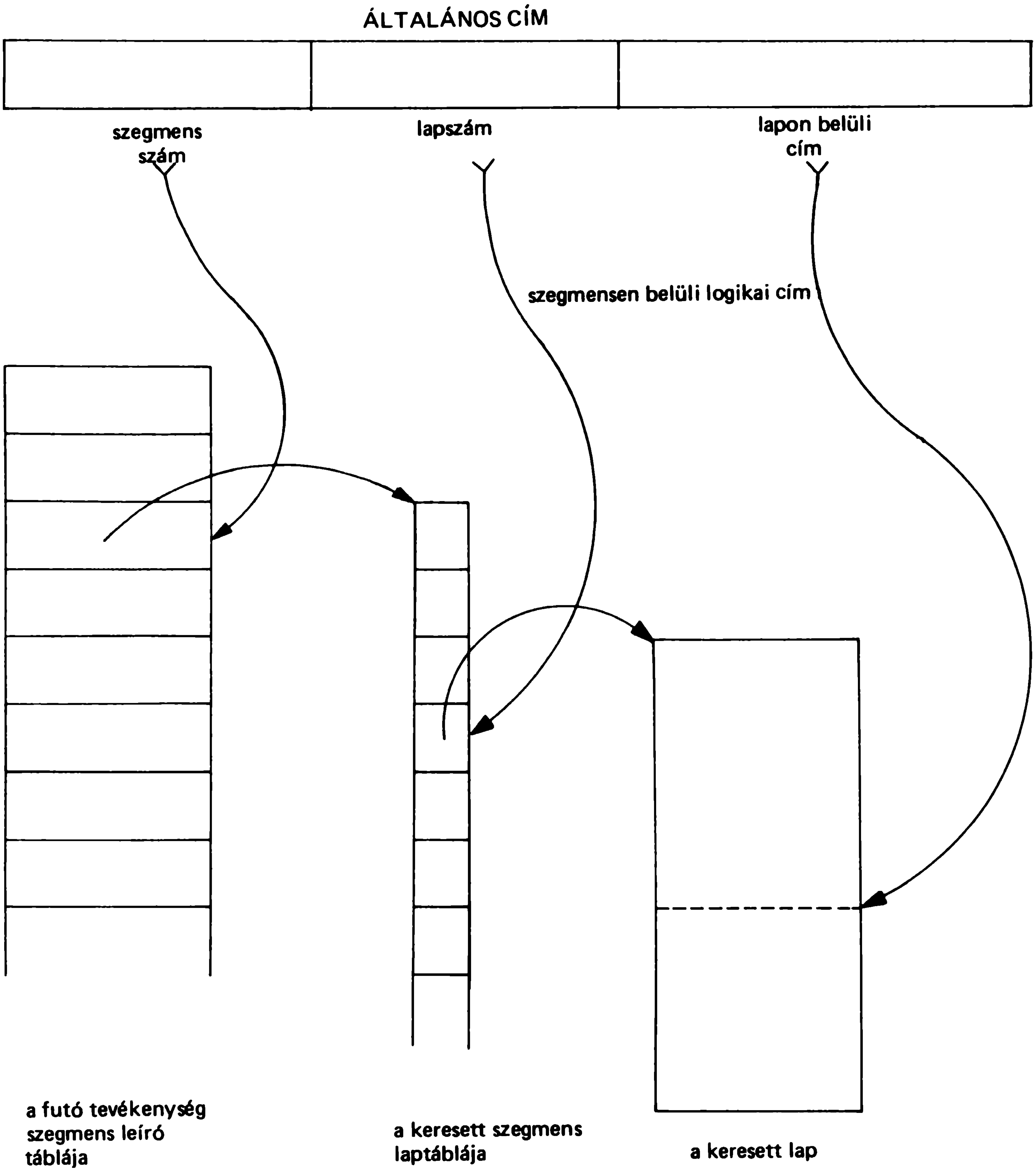
Egy program feladata, hogy futása során bizonyos memóriaterületek tartalmát egy algoritmusnak megfelelően átirja, módositsa vagy kitöltse. Egy adott program által használt memóriaterületnek két tipusa van. Az első tipus olyan információt tartalmaz, amely a számitógép központi egységét a kivánt algoritmus végrehajtásában vezérli. A második tipus olyan információt, melyet a program futása során elemez, módosit, átir. Az első tipust programnak, a másodikat adatmezőnek hivjuk. Általában egy program futása során nem csak az adatmezőt, hanem magát a programot is módosithatja, átirhatja. Nyilvánvaló azonban, hogy egy olyan program, amely közös software elemként kiván müködni, le kell, hogy mondjon erről a lehetőségről. Hiszen, kicsit eltulozva a dolgot, azt is képzelhetjük, hogy a program egy utasitásának végrehajtása után a rendszer átkapcsol a másik tevékenység végrehajtására (kicseréli a bázisregiszterek, illetve a laptábla tartalmát), és ugyanaz az utasitás most a másik tevékenység számára dolgozik. Ha az első tevékenység futása alatt ez a programelem megváltoztatja magát, az feltétlenül megzavarja a másik tevékenység kiszolgálását. Hasonló módon belátható, hogy a program nem változtathatja meg olyan adatmezőnek a tartalmát sem, amely mező a két tevékenység közös fizikai cimtartományában helyezkedik el. Ez gyakorlatilag annyit jelent, hogy a program adatmezejének a közös cimtartományon kivül, a programtól fizikailag elválasztva kell elhelyezkednie. Az ilyen szabályoknak eleget tevő programot többszörös (reentrant), vagy tiszta programnak (pure code) szokás nevezni. Alkalmazása egyrészt nagymértékü memória megtakaritást tesz lehetővé, másrészt a lapváltások számát is csökkenti. Ugyanis a rendszerek a közösen használható software-elemek lapjait lehetőleg nem cserélik, s igy több rendszer bizonyos számu lapja állandóan a memóriában van.

### *2.3.2. Szegmentálás*

A laptechnika előnyeinek elemzésénél (2.2.2.2. pont) láttuk, hogy ezzel a hardware megoldással kezünkbe került a reentrant software széleskörü használatának elvi lehetősége. Mindössze csak az okoz nehézséget, hogy a reentrant rutinok ugyanolyan formában szolgálják a különböző tevékenységeket, tehát a logikai cimtartomány meghatározott helyén keresik a manipulálandó adattömböt. Ha ezek a software elemek maguk is reentrant elemeket vesznek igénybe, akkor a hivó és a hivott egymáshoz képesti helyzete is rögzitődött, és az ilyenfajta programrészek kiterjedt használata hamarosan azt eredményezi, hogy a teljes logikai cimtartományt - mint egy óriási mozaikot - meg kell tervezni. Tulajdonképpen egy általános érvényü tevékenységet kell irni és a különböző konkrét esetekben ezen általánositott tevékenység meghatározott részeit kellene futtatni. Az első pillanatban ez nem tünik tulzott problémának, hiszen a nagy logikai cimtartománynak egy hasonlóképpen nagy, virtuális fizikai cimtartományt tudunk megfeleltetni, s ennek ellenére a tevékenység által használt valóságos memóriaterület kicsiny - szélsőséges esetben egyetlen lap - lehet. Az óriási cimtartomány kifejezéséhez azonban igen nagyszámu cimbit szükséges. A cimbitek csökkentésére irányuló törekvés végighuzódik a számitógép fejlődésének történetén. Ezen tulmenően megoldhatatlan probléma az is, hogy egy ilyen rendszer felépitésénél meg kell tervezni, hogy a rendszert hány program fogja alkotni és ezeknek milyen méretük lesz. Ha a részelemek közül egy is tullépi a számára engedélyezett keretet, az összes többi elemet ujra kell irni. Hasonló a helyzet, ha valamelyik részelemben egy belépési pont, vagy egy hivatkozási cim - a részelem módositása, tökéletesitése folytán - megváltozik. A lehetetlennek tünő feladat a következő: olyan hardware-software mechanizmust kell konstruálni, amely továbbra is lehetővé teszi a reentrant rutinok kiterjedt használatát, de

- nem igényel sok cimbitet,
- és lehetővé teszi az egyes software elemek szimbolikus egymásra hivatkozását.

Ez utóbbi annyit jelent, hogy egy program irója névvel (cimkével) hivatkozhat egy másik elem adott pontjára, és a program futása idején ez a szimbólikus hivatkozás automatikusan alakul át a megfelelő logikai cimmé. Ebben az esetben egy programelem módositása esetén a többi software elemet sem átirni, sem ujra forditani, de még ujra összeépiteni (linkelni) sem lenne szükséges. Ezt a lehetetlennek látszó feladatot a szegmentálás elvének kidolgozásával oldották meg. Ennek az elvnek sok konkrét implementációja ismeretes, de mindmáig a legtökéletesebb a GE-645-ös tipusu számitógépen megvalósitott MULTICS rendszer. A továbbiakban ezt a megoldást ismertetjük. /Irod. 7./


ÁLTALÁNOS CÍM
szegmens szám
lapszám
lapon belüli cím
szegmensen belüli logikai cím
a futó tevékenység szegmens leíró táblája
a keresett szegmens laptáblája
a keresett lap

*6. ábra*
*A fizikai cim meghatározása az általános cim alapjan*

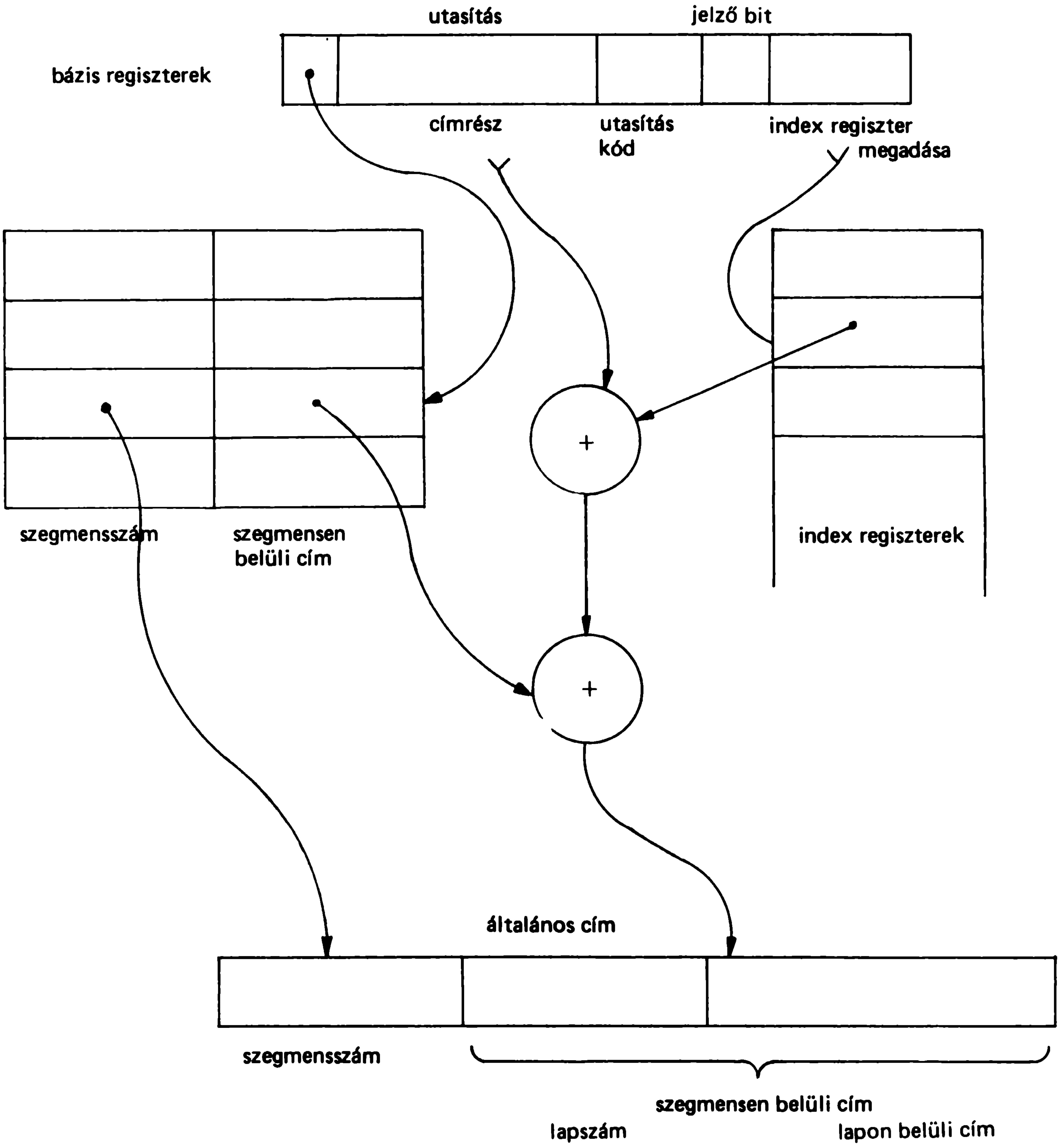
Szegmentált rendszernél minden tevékenység szegmensekből áll. A szegmens egy önálló logikai cimtartománnyal rendelkező, névvel ellátott egység. Ha egy szegmens önmagán belüli elemekre (utasításra vagy adatra) hivatkozik, akkor ezt az eddig megismert módon teszi, miközben ezt a logikai cimtartományt a hardware laptechnika segitségével leképezi a fizikai cimtartományra. Ha egy szegmens más szegmensen belüli pontot cimez, akkor ezt egy általánositott cimmel teszi. Az általánositott cim (Generalized Address) két részből áll. Az első rész a szegmenset azonositja, a második pedig a cimzett szegmens logikai cimtartományát cimzi. Igy az általánositott cimnek csak annyi bittel kell nagyobbnak lennie a maximális logikai cimtartomány cimzéséhez szükséges bitszámnál, ahány a rendszer által maximálisan engedélyezett szegmensszám kifejezéséhez elegendő. Természetesen rögtön felmerül a kérdés, hogy hogyan határozza meg a hardware az általános cimből a konkrét fizikai cimet. A megoldás a következő: minden tevékenységnek van egy szegmensleiró táblája (Segment Descriptor Table), melynek az egymást követő pozicióira hivatkozik az általános cim első része - a szegmensszám.

Az egyes táblapoziciók a keresett szegmens fizikai elhelyezését leiró táblázat - a laptábla - fizikai kezdőcimét adják meg. Igy a hardware a szegmensszám alapján előkeresi a hivatkozott szegmens laptábláját, és ennek segitségével meghatározza az általános cim második részének - a szegmensen belüli logikai cimnek megfelelő fizikai cimet (6. ábra).

Természetesen minden tevékenységnek önálló szegmensleiró táblája van. Ez a tábla valahol a memóriában van elhelyezve. A hardware-nek ismernie kell az éppen futó tevékenységhez tartozó szegmensleiró tábla kezdőcimét. Ennek tárolására egy külön regiszter: a szegmensleiró tábla bázisregisztere (Descriptor Base Register) szolgál. Ha a supervisor egy másik tevékenységet indit el, akkor ennek a regiszternek a tartalmát megfelelőképpen módositja. Az eddig ismertetett megoldással a kitüzött feladat első része - a logikai cimbitek számának csökkentése, illetve az egyes szegmensek hosszának kötetlen kialakitása - megoldott.

Nem látszik azonban lehetőség a különböző szegmensek, illetve a szegmensen belüli különböző pontok szimbolikus cimzésére. A programozónak szegmensszámot és konkrét szegmensen belüli, logikai cimet kell megadnia, ezeket azonban nem ismeri. A rendszer lehetőséget ad arra, hogy 4 különböző, a tevékenység futása szempontjából döntő jelentőségü logikai pontot, illetve annak környezetét, a programozó közvetlenül cimezni tudja, anélkül, hogy szegmensszámot, illetve szegmensen belüli cimet ismernie kellene. Erre a célra a GE-465-ös számitógép 4 bázisregiszterrel rendelkezik (7. ábra).

Ha az utasitásban levő jelzőbit értéke 1, akkor a gép a cimrész felső bitjeit másképp értelmezi, mint egyébkor. Ilyenkor ez a rész hivatkozik valamelyik bázisregiszterre. E regiszter felső részében


utasítás
jelző bit
bázis regiszterek
címrész
utasítás kód
index regiszter megadása
+
+
szegmensszám
szegmensen belüli cím
index regiszterek
általános cím
szegmensszám
szegmensen belüli cím
lapszám
lapon belüli cím

*7. ábra*
*Az általános cim meghatározása a bázisregisztereken keresztül*

elhelyezett szegmensszám lesz az általános cim szegmensszám része. A szegmensen belüli cim pedig az utasitásban megadott cim, az indexregiszterben megadott érték, és a bázisregiszter alsó részében talált alapcim összege lesz. E módszerrel lehetővé vált négy különböző logikai pont rendszerkonvención alapuló cimzése. Azonban, ha talán nem is olyan sürün, mint ezeket a pontokat, de sok egyéb helyet is kell időnként cimezni. A tárgyalt számitógép indirekt cimzési technikája erre lehetőséget ad. Ha egy utasitásban indirekt cimzés jelzése van, akkor a hardware az utasitásban megadott cimről *két* egymást követő szót olvas ki. Ez a dupla szó az indirekt cimszó. Értelmezése, bizonyos jelzőbitek beállitásától függően, többféle módon történik:

1) Az operandust ugyanazon szegmensből kell venni, ahol az indirekt cimszó volt. A szegmensen belüli logikai cimet a dupla szó első tagja tartalmazza.

2) Az operandust abból a szegmensből kell venni, amelynek száma az indirekt pár első tagjában volt, a szegmensen belüli logikai cimet az indirekt pár második tagja tartalmazza.

3) Az operandust abból a szegmensből kell venni, amelynek száma az indirekt pár első tagjában kijelölt bázisregiszterben van, a szegmensen belüli logikai cimet az indirekt pár második tagja tartalmazza.

4) Az operandust abból a szegmensből kell venni, melynek szimbolikus cime (neve) az indirekt pár első tagja, mig a szegmensen belüli cim szimbolikus megadása, a cimet azonositó cimke (label) a pár második tagjában van.

A fenti módokon cimzett operandus egy ujabb indirekt pár is lehet. Az első indirekt párt a program vagy saját logikai cimtartományán belül tartalmazza, vagy pedig a bázisregisztereken keresztül közvetlenül cimzi más szegmensekben. Ezután az indirekt cimzés különböző szinteken és különböző szegmenseken futhat keresztül, mig a tulajdonképpeni operandus cimét meg nem kapjuk. A szimbolikus cimzés szempontjából az indirekt cimzés negyedik fajtája fontos. Ez a cimzési forma programmegszakitást okoz. A supervisor minden tevékenységgel kapcsolatban külön és együttesen is adminisztrálja a használt szegmensneveket. Ha az indirekt cimben szimbolikus hivatkozást talál, akkor megvizsgálja a tevékenységhez tartozó "ismert szegmensek táblázatát".

Ha ott megtalálja a keresett nevet, akkor a szegmens már szerepelt a tevékenység munkája során, van poziciója a szegmens leiró táblában, ennek a poziciónak a számát - a szegmensszámot - visszairja az indirekt dupla szó első tagjába. Ha a keresett szegmensnév nem szerepel a tevékenységhez tartozó "ismert szegmensek táblázatában", akkor még szerepelhet a rendszer összes létező tevékenységéhez tartozó ismert szegmensek táblázatában. Ha ez a helyzet, akkor a szegmenset a supervisor hozzárendeli ehhez az utóbbi tevékenységhez is. Laptáblájának cimét beirja a tevékenység szegmens leiró táblájának következő üres poziciójába, és e poziciónak a számát, minthogy a tárgyalt tevékenység szempontjából ez a szegmens hivatkozási száma, beirja az indirekt cimszó első tagjába. Ezután a szegmensen belüli szimbolikus hivatkozás feldolgozása következik.

Minden szegmensnek - a rendszer által ismert helyen - van egy tartománya, amely tartalmazza a szegmensen belüli definiált hivatkozási pontok nevét és az ehhez tartozó logikai cimet. A supervisor kiemelve innen a megfelelő logikai cimet, elhelyezi ezt az indirekt cimszó második tagjában. Ezután a tevékenység tovább futhat, és ha *ugyanezen* indirekt dupla szón keresztül cimzi a más szegmensben lévő hivatkozási pontot, akkor a cimlánc - megszakitás nélkül - közvetlenül feldolgozódik. Ezzel megoldódik a szegmens más szegmenseket érintő cimzése.

Van azonban még egy probléma. Ha az előbb leirt indirekt cimszó a szegmens logikai cimtartományán belül volt, azaz a szegmenshez tartozott, akkor a szegmens egy eleme *megváltozott*, ez pedig egy reentrant software elemnél megengedhetetlen. A szimbolikus kifelé cimzést végző, indirekt szópárok nem tartozhatnak tehát a szegmenshez. A probléma megoldására létrehozták a szegmens kapcsoló szegmensét (Linkage Segment). Minden szegmenshez tartozik *legalább egy* kapcsolószegmens, amely a kifelé történő cimzés indirekt szavait tartalmazza. Ha egy szegmens több tevékenységnek része ,akkor több kapcsolószegmenset kell létesiteni, hiszen ugyanazoknak a szimbolikus szegmensneveknek egy másik tevékenységnél más szegmensszámok fognak megfelelni, ugyanis minden szegmensleiró tábla más.

A kapcsoló szegmensek létesitését a supervisor végzi el akkor, amikor egy szegmenset egy tevékenység számára ismertté tesz. Már csak egy kérdés maradt nyitva: honnan tudja egy szegmens, hogy hol van a kapcsoló szegmense, hiszen kifelé cimezni csak a kapcsolószegmensen keresztül tud, igy kapcsoló szegmensét is csak a kapcsoló szegmensen keresztül tudja elérni. Többek között ennek a problémának a megoldására vezették be a bázisregisztereket (7. ábra). A második regiszterben mindig az éppen müködő szegmenshez tartozó adott tevékenységen belül érvényes kapcsolószegmens kezdőcime van.

Ha a rendszer a vezérlést egy másik tevékenységnek adja át, akkor lecseréli a négy bázisregiszter tartalmát is. Ha a vezérlést történetesen ugyanaz a szegmens kapja is meg, amely eddig dolgozott, akkor is második bázisregisztere most már egy másik kapcsoló szegmensre fog mutatni.

A fentiekben hézagosan ismertettük a MULTICS rendszer szegmenskezelő rendszerét, melyet azért dolgoztak ki, hogy a reentrant rutinok kiterjedt használatából adódó előnyöket az egyszerü megoldás hátrányai nélkül élvezhessék. Azonban talán az érintőleges bemutatásból is látható, hogy a rendszer teljes bonyolultságu kiépitése annyi hardware és software problémával, illetve költségtöbblettel jár, hogy a gyakorlati hátrányok vetekednek az elvi előnyökkel. Ez okból érthető, hogy bár a Burroughs B-5000 számitógép óta a Californiai Egyetemen kidolgozott SCC 6700-as rendszer mellett az IBM 360/67, a 370/VS és az RCA 70/46 is lehetővé teszi a szegmentálást, de jóval több megkötéssel, mint azt a MULTICS tette.

### 2.3.3. *Átlapolásos (Overlay) szerkezet*

Az előző pontokban leirt megoldások a programok memóriaigényét automatikusan, a programozó segitsége nélkül csökkentik. Tehet a helyigény csökkentéséért valamit azonban a programozó is.

A programozói memóriagazdálkodás eszköze az overlay szerkezet alkalmazása. Ez annyit jelent, hogy a programozó szétválasztja programjának azon funkcióit, amelyekre nem egyidőben van szükség, és ezeket a funkciókat különálló programszakaszokkal hajtja végre. Ezek a programszakaszok ezután a memória azonos területére töltődnek be, egyik a másik után. Az egyes programszakaszok betöltéséért a programozó a felelős, bár modern forditóprogramok már automatikus overlay szerkezetkezelést is lehetővé tesznek.

## 2.4. *File-rendszer kezelése*

### 2.4.1. *A file-rendszerrel szemben támasztott követelmények*

Mint ezt eddig is sokszor hangoztattuk, a terminálnál ülő felhasználónak bármikor joga van információt megőrzésre a rendszernek átadni, illetve ezt az információt később - akár évekkel később is - visszakérni, módositani és ujratárolni stb. A file névvel ellátott, szerkezettel biró adatstruktura. Ezeket a file-okat a rendszer egy olyan háttértáron tartja, amely lehetővé teszi, hogy több felhasználó szimultán file-kezelését biztositani lehessen. Ilyen eszköz a mágnesdob és a mágneslemez. Mindkét tárolóeszköznek közös tulajdonsága, hogy tárolási tartománya tetszés szerint felosztható, és bármelyik file bármelyik rekordja után - viszonylag kis pozicionálási idő után - bármelyik másik file bármelyik rekordja elérhető. A felhasználónak, mielőtt egy uj file-t venne használatba, ennek létesitését kérnie kell a rendszertől. A kérést a felhasználó és a vezérlősoftware közötti kommunikáció céljára létesitett vezérlőnyelv (command language) segitségével kell közölni. A kérésnek tartalmaznia kell a file nevét, és a file hosszát. Az utóbbira nyilvánvalóan azért van szükség, hogy a rendszer megfelelő területet foglaljon le a háttértárolón. Előfordulhat, hogy a háttértárolókon már nincs hely, ebben az esetben a rendszer kénytelen a file felvétele iránti kérelmet elutasitani (pl. SYSTEM OVERLOAD - A rendszer tulterhelődött - üzenettel). Nagyobb rendszereknél (pl. UNIVAC 1100) nem kell a kért file méretét megadni. Ez természetesen további nehézséget jelent a rendszer szempontjából. Mindenesetre meg kell adni a file nevét, s minden alkalommal, amikor a felhasználó ezt a file-t kéri, a rendszer ezt a név alapján tudja azonositani. Természetesen egy felhasználónak több file-ja lehet, de ezeknek mind különböző nevüeknek kell lenniük. A rendszernek tehát felhasználónként (minden felhasználónak egyedi azonositója van) egy katalógust kell karbantartania, melyben a szereplő file-ok nevei, azoknak fizikai elhelyezkedésére vonatkozó információ kell szerepeljen. Napjaink időosztásos rendszereinek a felhasználók szempontjából igen vonzó tulajdonsága, hogy más felhasználók file-jait is használhatják. A rendszernek lehetővé kell tennie tehát azt is, hogy egy felhasználó bizonyos file-jait (csak azokat, amelyeket engedélyezett) mások igénybevehessék. Ennek szokásos megoldása a következő: a felhasználókat csoportokba soroljuk. Minden csoportnak van egy csoport file-tára. Az itt elhelyezett file-okat a csoport tagjai használhatják, bár megkötésekkel. Nyilvánvalóan kell lenni legalább egy állomásnak, ahonnan a csoport közös file-jait módositani, törölni lehet. Más állomások a közös file-okat csak olvashatják, néhányan esetleg módosithatják, de nem törölhetik stb.

Ezzel kapcsolatban felmerül egy további probléma. A rendszer a jogokat az állomás azonositója alapján birálja el. Biztositani kell, hogy rosszindulatból vagy tévedésből egy állomás ne szerepelhessen más néven, mint amihez joga van. Ha a számitógéphez csatolt terminálok állandó és egyedi vonalra vannak kötve, akkor az azonositás történhet a vonal alapján. Postai vonalon jelentkező állomásoknál az azonositót, mint input adatot kell a rendszernek bekérnie. A biztonság fokozása érdekében némely rendszer a felhasználótól jelszó beadását is megköveteli, s ha ez nem egyezik a kérttel, akkor a terminál kiszolgálását elutasitja. Egyes rendszerek még arról is gondoskodnak, hogy a jelszó ne lehessen a jogosulatlanok számára megszerezhető. Ugyanis irómüves terminálnál a beütött jelszó megmarad a berendezés papirján. Szokásos dolog a jelszó beadása idejére az irómüvet kikapcsolni, vagy megengedni, hogy a jelszó olyan karakterekből álljon, amelyeknek nincs nyomtatott megjelenése (pl. a táviróberendezések vezérlőbillentyüinek kódja), vagy a rendszer az irómüvet visszalépteti a jelszó elejére és pl. csupa X betüvel a jelszót felirja.

Nagyobb rendszereknél a csoport közös file tárán kivül, még magasabb szintü közösségek is lehetnek, sőt szokott lenni egy filetár, amelyet minden felhasználó használhat.

### *2.4.2. A file-ok tipusai*

Az időosztásos rendszerek kétféle tipusu file-t szoktak kezelni. Az első tipusba tartoznak a program file-ok. Ezekben a file-okban forrásnyelvi szintü (source code), félig forditott, vagy teljesen leforditott programok vagy programelemek illetve szubrutinok lehetnek. Lehet a felhasználó csak arra jogosult, hogy ezeket a programelemeket a programjához csatolja, anélkül, hogy tartalmát kiirathatná, vagy arra, hogy programjához csatolja és kiirathassa, vagy pedig arra is, hogy a programot módositsa.

A második tipusba tartoznak az adatfile-ok. A jogosult felhasználónak ezeket tudnia kell elemenként olvasni. Lehetőséget kell adni a file rekordonként történő végigjárására (szekvenciális hozzáférés), de a tetszőleges sorrendü (random) olvasásra is. Bonyolultabb rendszerek lehetőséget adnak arra is, hogy a közösen használható file-okat egyidőben több felhasználó is olvashassa. Több joggal rendelkező felhasználó ezeket a file-okat módosithatja, bővitheti vagy meg is szüntetheti, de egyidőben csak egy felhasználó teheti ezt.

# 3.

# Az időosztásos üzemmód különleges szolgáltatásai

## *3.1. Interaktiv üzemmód, interaktiv forditóprogramok*

### *3.1.1. Az interaktivitás első foka - mindent a terminálról*

A kötegelt feldolgozás az alábbi lépéseket teszi szükségessé annak érdekében, hogy a programozó elkészitse programját:

a) a program megirása,
b) a megirt program adathordozóra lyukasztása,
c) az adathordozón lévő program leforditása, a szintaktikus elemzés elvégeztetése,
d) ha nincs szintaktikus hiba, akkor továbblépés a h) pontra,
e) a szintaktikus hiba megkeresése a forditóprogram jelzése alapján, a javitás elkészitése,
f) a javitás elvégeztetése a program adathordozóján,
g) visszatérés a c) pontra,
h) a program futtatása, a szemantikus vizsgálat,
j) a program outputjainak, illetve a rendszer hibajelzéseinek elemzése; ha a program hibátlanul lefutott, továbblépés m) pontra,
k) a programhiba megkeresése, és a javitás elkészitése,
l) visszatérés az f) pontra,
m) a program dokumentálása rendszeres futtatáshoz.

Ezek a lépések részben feleslegesek, részben kiszámithatatlan időigényüek, mert a végrehajtás ellenőrzése kivülesik a programozó hatáskörén. Az időosztásos rendszer lehetővé teszi, hogy az összes feladatot a programozó végezhesse el, ezzel a lényegében felesleges lépések megszüntethetők. Hogy ez hogyan történik, azt nézzük végig egy konkrét példán. A következőkben bemutatjuk, hogy hogyan kell egy egyszerü programot irni FORTRAN nyelven interaktiv üzemmódban. (A bemutatott példában a programhibák javitásának menete is követhető.)

A felhasználónak először is termináljét kapcsolatba kell hoznia a rendszerrel, azután elvégezni az azonositási procedurát:

```
! BTM ON SIGMA 5

 10/02/'73 13 : 11
```

Először a rendszer jelentkezik, és közli a dátumot és a pontos időt (az aláhuzás azt jelzi, hogy ezeket a karaktereket a számitógép iratja ki).

```
! LOGIN : ACC1, SDS, 6005
ID = C
```

Ezután a felhasználó azonositja magát, melyet a rendszer elfogad.

```
! TABS  7,72
```

A felhasználó beállitja a TABULÁTORT a FORTRAN nyelv szintaxisának megfelelően.

```
! EDIT
* BUILD FTEST
```

A felhasználó közli, hogy programot akar késziteni, méghozzá uj programot, melyet FTEST nevü file-ban óhajt őrizni.

```
1.000   C   ATLAG SZAMITO PROGRAM
2.000   C   OT ELEM ATLAGAT SZAMITJA
3.000       READ (6,10) A, B, C, D, E
4.000       S -  (A+B+C+D+E)   /5.0
5.000       WRITE (6,20)  S
6.000       FORMAT (5F 5.2)
7.000       FORMAT (3HS= , F6.1)
8.000       END
9.000

* END
```

Befejeződött a program irása, a felhasználó ezt közli a rendszerrel.

```
! ASSIGN M :SI,  (FILE, FTEST)
! FORTRAN
```

A felhasználó közli, hogy a forditó program inputja (SI = SOURCE INPUT) a FTEST nevü file legyen, és meghivja a FORTRAN forditóprogramot.

```
OPTIONS:   BO
```

Megadva a fordító programnak a kért paramétert (Binary Output) elkezdődik a fordítása:

```
4:            S - (A+B+C+D+E)  /5.0
                ↑
****          SYNTAX
****          LABEL ERRORS
**            END OF COMPILATION **
```

A fordítás végén a fordítóprogram jelzi, hogy a 4. sorban formai hiba van, ezenkivül cimke hibák vannak. (A függőleges vonal azt jelzi, hogy ezt a szöveget a számitógép irja ki.)

```
!   EDIT

*   EDIT FTEST

*   IN 4

    4.000         S = (A+B+C+D+E) /5.0

*   IN 6

    6.000   10    FORMAT (5F5.2)

*   IN 7

    7.000   20    FORMAT (3HS= , F6.1)

*   END
```

A felhasználó ujra meghivja a szerkesztő programot, közli a változtatandó file nevét és a sorok számát, amelyet változtatni kiván.

Ezután sorra beirja a megváltozott FORTRAN utasitásokat.

```
!   FORTRAN

OPTIONS:  BO

**  END OF COMPILATION **
```

A fordítóprogram most nem talált hibát.

```
! LOAD
ELEMENT FILES:
OPTIONS:
F: 6
F;
```

A felhasználó betölti az imént leforditott programot. A töltő program bekéri a paramétereket. Ebben az esetben meg kell adni, hogy a READ és a WRITE utasitásban szereplő 6-os periféria a terminál.

```
SEV. LEV. = 0
XEQ ? Y
```

A betöltés sikerült, utasitjuk a rendszert, hogy inditsa el a programot.

```
:    2.    3.    4.    5.    6.
S =  4.0
STOP
USER EXIT
stb.
```

Mint a fentiekből látható, a programkészités menete most:

a) a program megirása,
b) a program leforditása, szintaktikus ellenőrzés,
c) a program javitása, vagy továbblépés a d) pontra,
d) a program futtatása,
e) a program elemzése az outputok és a rendszer jelzései alapján, ha hiba van visszatérés a c) pontra.

### *3.1.2. Az interaktivitás második foka - inkrementális forditóprogram*

Az előbbi pontban ismertetett géphasználati mód a felhasználó számára sokkalta kényelmesebb, mint a kötegelt feldolgozás kötelmei szerinti munka. Az ilyen rendszerek használata során azonban kiderült, hogy további lépések összevonása is célszerü. Például a FORTRAN nyelvnél általában egy utasitás önmagában szintaktikus egész. Ha az utasitás leirásakor a rendszer nemcsak egyszerüen átveszi a leirt sort, hanem szintaktikusan ellenőrzi is, akkor a formai hibák kijavitása azonnal megtörténhet. Továbbá kiderült, hogy a terminálnál ülő felhasználó nem szeret és nem tud gyorsan billentyüzni. Vizsgálat alá vették a forditóprogramok szintaxisát a terminálok igényének megfelelően. Többek között feleslegesnek bizonyultak a FORTRAN numerikus utasitáscimkéi, hiszen az egyes programsorokat ugy is be kell sorszámozni az egyszerü javithatóság kedvéért. Tul körülményesnek bizonyult a FORTRAN READ, illetve WRITE utasitáshoz kapcsolandó FORMAT utasitás is, hiszen a terminálon történő adatmegjelenités eleve megköti az output formátumát. Revizió alá kellett venni a FORTRAN file kezelési rendszerét is, hiszen itt is uj szempontok merültek fel (ld. 2.4.2. pontot). További szempont az is, hogy a nyelv szintaxisát minél könnyebben lehessen elsajátitani. A fentieknek megfelelő szempontoknak megfelelően igen sok uj terminál-orientált programnyelv alakult ki. Ezek közül egy a FORTRAN-hoz igen hasonló BASIC (Beginners All-purpose Symbolic Instruction Code), melynek kifejlesztésénél - mint ezt a nyelv neve is mutatja - különösen a kezdő programozók érdekét tartották szem előtt.

Nézzük meg, hogy az előbbi átlagszámitó program hogyan készithető el a HEWLETT PACKARD cég TSB rendszerén BASIC program nyelven:

```
HELLO - H200, SECRET
READY
```

A felhasználónak természetesen kapcsolatba kell lépnie a rendszerrel, közölnie kell azonositóját és a jelszót. Ha a TSB elfogadja jelentkezését, akkor a READY üzenettel további feladatra várakozik.

```
10   INPUT     A,B,C,D,E
```

A felhasználó elkezdi programját irni. Nem kell azonban ezt a szerkesztő program meghivásával kezdenie (v.ö. előző pont), a rendszer felismeri a BASIC utasitásokat a sorszámról, amelyeket most a felhasználónak kell megadnia.

```
30   PRIMT    S
ERROR
```

A következő utasitás hibásan lett leirva, a rendszer ezt azonnal jelzi. A felhasználó idejének megtakaritása céljából ezt a jelzést egy rövid üzenettel teszi. Ha azonban ennyi jelzés a program irójának nem elegendő, akkor az ERROR üzenet után kettőspontot beütve részletes hibajelzést kap.

```
ERROR : NO STATEMENT TYPE FOUND
```

A hiba kijavitásához egyszerüen meg kell ismételni az utasitást helyesen.

```
30 PRINT S

20 LET S= (A+B+C+D+E) /5

50 END
```

A felhasználó programjának sorait tetszésszerinti sorrendben közölheti a rendszerrel, a program a növekvő utasitásszámok, nem a közlés sorrendje szerint lesz végrehajtva.

```
RUN
```

A felhasználó futtatni óhajtja a most irt programot:

```
? 95.6,87.3,80.5,90,82.8
```

A program elindult és elérve az INPUT utasitást, várja az adatokat. (A BASIC programozónak nem kell az integer és a real ábrázolás közötti különbséggel törődnie.)

```
87.24

DONE
```

A program kiirja az eredményt, s a rendszer közli, hogy a program lefutott.

```
BYE

003 MINUTES OF TERMINAL TIME
```

Miután a programozó elbucsuzott a rendszertől, megkapja a felhasznált (számlázási alapként szolgáló) időről az információt.

Látható, hogy a forditóprogram utasitásonként, programegységenként, inkrementumonként fordit. Az ilyen forditóprogramokat - inkrementális forditónak nevezzük. Használatuk lehetővé teszi, hogy a programirásnál az irás, szintaktikus ellenőrzés, forditás egyetlen menetben történhessen.

A programkészités menete most az alábbi:

a) a program megirása, szintaktikus ellenőrzése, forditása,
b) a program futtatása,
c) a program ellenőrzése az outputok és a rendszer jelzései alapján, ha hiba van visszatérés az a) pontra.

### *3.1.3. Az interaktivitás harmadik foka - interpretiv forditóprogram*

Az előző paragrafusban ismertetett programkészitési mód nagymértékben egyszerüsiti a programirást, a programpróba azonban pontosan olyan komplikált, mint a kötegelt feldolgozásnál. Pedig ezen a területen is nagyon ráfér a programozókra egy kis segitség. Aki próbálta, az a megmondhatója, milyen bonyolult feladat egy nagyméretü, sok rutint használó program első futásánál az outputból megállapitani, hogy hol lehet a hiba. A hardware évtizedében dolgozó programozó dolgát rengeteg hardware funkció segitette: futás közben bármikor megállithatta a programot, vagy különböző okok szerint megállittathatta (pl. ha az utasitás számláló bizonyos értéket felvesz, vagy ha a program bizonyos cimre irni akar, vagy ha bizonyos utasitáskód kerülne végrehajtásra stb.). Ilyenkor - ha ismerte a gép bináris utasitás, illetve számábrázolását - megvizsgálhatta változóinak értékét, sőt módosithatta azokat, és a programot továbbindithatta.

Ezt az azóta is sokak által visszasóvárgott lehetőséget kinálják az időosztásos rendszerek legmodernebb tipusu forditó programjai, az interpretiv forditó programok. A programozó lehetőségeit a UNIVAC 1108-as számitógépre kidolgozott CFOR (CONVERSATIONAL FORTRAN) nyelven irt példák segitségével mutatjuk be:

Lehetőség van arra, hogy a programot utasitásonként, ugy, ahogy leirjuk, le is futtassuk. Természetesen ez esetben ügyelni kell arra, hogy a jobboldali változók definiáltak legyenek.

```
100.  READY   DIMENSION  A(100)
```

A felhasználó elkezdi egy FORTRAN program irását. A sorokat a CFOR automatikusan számozza.

```
101.  READY   EXR
```

A programot iró ugy rendelkezik, hogy minden ezután következő végrehajtható FORTRAN utasitást azonnal végre is kell hajtani.

```
 101.  READY  20  FORMAT  (F8.3)
 102.  READY      READ    (2,20)  B,C
-102.  READY
```

Az először leirt FORMAT nem végrehajtható utasitás (azért irtuk először, mert a következő READ igényelni fogja), a READ utasitást azonban rögtön végre is hajtja a gép. A sor elején megjelenő (-) jel inputot kér, a sorszám pedig az eredeti program megfelelő utasitásának sorszáma.

Miután a felhasználó a megfelelő értéket a terminálon beadta, a CFOR várja a következő utasitást.

```
 103.   READY    D = 20
 104.   READY    F = 20 - B - C
 -104   YIELD    F = valamilyen érték
```

A 103. utasitás egyszerü értékadó utasitás volt, a 104-es azonban már nem, és ezért a CFOR az utasitás végrehajtása után tájékoztat F uj értékéről. Tegyük fel, hogy ebben a pillanatban észrevesszük, hogy a 104-es utasitás tartalmilag nem jó. Az utasitást rögtön javithatjuk.

```
  105.   READY   UPDATE
* 105.   UPD    - 104, 104
  104.   UPD     F = 25 - B - C
  104.1  UPD    ∇OFF (UPDATE)
  105.   READY
```

A 104-es sor után még további sorokat is beszurhattunk volna az UPDATE funkcióval, ezt jelzi a 104.1 alátöréses sorszám, azonban mi itt befejeztük a beszurást, a rendszer visszatért arra a pontra, ahol az utasitások irását abbahagytuk. Az UPDATE funkcióval beszurt sorokat az a funkció lezárása után (OFF) a CFOR szintaktikailag ellenőrzi, de nem hajtja végre. Ha most elölről szeretnénk programrészletüket futtatni, akkor

```
 105.   READY   ∇BEGIN 103/104
```

utasitással tehetjük.

A BEGIN utasitással teljes programot, programrészletet vagy szubrutint hajtathatunk végre. Ha szubrutint hajtunk végre, akkor a hivási paraméterlistát is össze tudjuk állitani

```
 100.   READY    BEGIN (RTN, B, I, J,  100.8)
```

formában. Itt utasitottuk a rendszert, hogy futtassa le az RTN azonositóju szubrutint a B, I, J változókból és 100.8 értékü konstansból álló paraméterlistával.

A futó programot futás közben megszakithatjuk vagy megszakittathatjuk. Megszakitásra a terminálon elhelyezett különleges billentyü szolgál. Megszakittatásra a CFOR dinamikus teszt utasitásainak egy része szolgál. Például GUARD sorszám/sorszám utasitással programmegszakitást lehet kérni egy programtartomány utasitásainak végrehajtása előtt. A LIMIT utasitás megszakitást okoz, ha a megadott változó értéke megfelel egy relációnak. A KEYIN utasitással pedig egyetlen utasitást lehet megjelölni, melynek végrehajtása *előtt* a program futása megszakad. A megszakitás után a programozó a RESUME utasitással indithatja tovább a programot a megszakitási pontról.

További lehetőségek:

A dinamikus tesztutasitások másik csoportja lehetővé teszi a program müködésének követését, de interrupt nélkül. Például a TRACE utasitás a megadott változó értékének megváltozásáról, a TRAP utasitás a vezérlésátadásokról (GO TO), a TRAIL pedig a függvényutasitások, függvényrutinok (DEFINE Function), belső és külső függvények hivásáról informál.

A statikus tesztutasitások a program megállitása vagy lefutása után adnak információt: pl. ki lehet iratni változók értékét, adott utasitás végrehajtásának számát, vagy a futás során "nem szereplő" utasitások, illetve változók azonositóját.

Az eddig ismertetett funkciók igen kényelmessé teszik a program kipróbálását. Ennek azonban ára van. Minthogy a forditóprogram futás közben értelmezi az utasitásokat, a futás meglehetősen lassu. A program tulajdonképpen utasitásonként a futás pillanatában értelmeződik - ezért hivják ezt a forditóprogramot interpretiv forditóprogramnak. Az ilyen rendszerek általában lehetővé teszik, hogy a programot vagy programrészletet (végleges kipróbálása után) leforditsuk, és ebben a formában tároljuk. Ezt a formát később bármikor futtatni, vagy egy készülő programhoz kapcsolni lehet, azonban természetesen ez a programrész már nem futtatható az interaktiv szolgáltatásokkal.

A programkészités menete - az utóbb leirt szolgáltatások eredményeképpen - egyetlen menetben történhet:

- a program megirása, szintaktikus és szemantikus ellenőrzése, futtatása.

### *3.2. Időosztásos üzemü előfizetői rendszerek*

Futurológusok régóta jósolják, hogy a számitógép hamarosan éppugy bevonul mindennapi életünkbe, mint a telefon, vagy a televizió. Sokan, sokáig kétkedve fogadták ezt a véleményt, hiszen a számitógép olyan drága berendezés, és programozása olyan bonyolult. Az időosztásos rendszer alkalmazása megteremtette a lehetőséget arra, hogy egy nagy számitógépet, vagy akár számitógépek hálózatát nagyon-nagyon sok felhasználó használhasson. Az egy felhasználóra eső fenntartási költség alacsony lehet. Egy uj állomás bekapcsolásakor felmerülő beruházási költség pedig - a terminál költsége - hamarosan egy televiziós berendezés árkategóriájába fog esni. Ebben a fejezetben azt is bemutattuk, hogy a programozás is igen egyszerü ezeken a rendszereken. Általános tapasztalat szerint számitógépet sohasem látott, átlagos képzettségü ember a BASIC programnyelv alapjait egy hét alatt tökéletesen elsajátithatja. Természetesen ahhoz, hogy valaki az idõosztásos üzemmód segitségével egy számitógépet használjon, ahhoz még programozni sem kell tudni. A legtöbb embert érintõ problémára nyilvánvalóan programokat lehet kidolgozni, amelyet a többiek megvásárolnak, és mindössze a futtatáshoz kell némi képzettségre szert tenniük. Némivel tapasztaltabbak már vállalkozhatnak arra is, hogy ezeket a tipus programokat speciális igényeknek megfelelõen adaptálják.

A terminálok országos hálózata is elképzelhetõ, a postai vonalak már ma is rendelkezésre állanak, bár az automatikus távválasztásnak és a vonalak zavarmentességének még sokat kell fejlõdniük ahhoz, hogy tömeges adattávátvitelre alkalmasak legyenek.

# Irodalomjegyzék

/1/ Saltzer, I.H.: Traffic Control in a Multiplexed Computer System, Massachusetts Institute of Technology, 1966. Cambridge, Mass.

/2/ Watson, R.W.: Time-sharing System Design Concepts McGraw-Hill Book Company, 1970, New York.

/3/ CONTROL DATA CORPORATION: 7600 Computer System; SCOPE 70 General Information Manual, CDC, 1970.

/4/ Reiter, Allen: A Resource-allocation Scheme for Multi-user On-line Operation of a Small Computer, AFIPS CONF. PROC. vol. 30. Thompson - Acad. Pr. 1967. Washington - London.

/5/ Rehmann, Sandra L.: A Simulation Study of Resource Management in a Time-sharing System, AFIPS CONF. PROC. vol. 33, Thompson Books, 1968, Washington.

/6/ Varian, L.C., Coffman: An Empirical Study of the Behavior of Programs in a Paging Environment, Proc. ACM Symp. Operating Systems, Thompson - Acad. Pr. 1967. Washington - London.

/7/ Daley, R.C., Dennis: Virtual Memory Processes, and Sharing in Multics, Commun. ACM, vol.11, 1968. no. 5. p. 306.

# Tárgymutató

A

adás 154, 156
adás vége jel (EOT) 123, 154
adatállomás 95
adatállomás, számitógépes 95
adatátvitel, analóg 106
adatátvitel, digitális 107
adatátvitel, vezérlési eljárások 147-159
adatátviteli hálózat 95, 114-133
adatátviteli kisszámitógép 192-199
adatátviteli kódok 72
adatátviteli programcsomag 187-191
adatátviteli sebesség 99-100
adatátviteli software 98
adatátviteli számitógép 87
adatátviteli vezérlés 98
adatátviteli vezérlő egység 96
adatátviteli vezérlő karakter 137, 147, 151, 154, 161
adatcsatorna 95
adatgyüjtő rendszer 92
adathálózat, független 132-133
adathálózat, nyilvános 128-131, 163
adatkezelés (data management) 22, 37-40
adatkoncentrálás 113, 197-199
adatszétosztó rendszer 92
adatvégállomás 95, 96
adatvonal 95, 99-113
adatvonal, analóg 106
adatvonal, digitális 107
adatvonal koncentrátor 109, 110, 112-113, 149
adatvonal multiplexer 109-112
alállomás 107
alcsatorna 172
állapotszó (PSW) 176, 183
amplitudó moduláció 106
aránykód 142
aszinkron átvitel (start-stop átvitel) 102, 104
aszinkron átvitel vezérlése 150-151
asszociativ tár 58, 81
áthelyezhetőség (relokálhatóság) 71, 177-178, 223
átlapolás (overlay) 241
átlapolás tár cimzésnél (overlapping) 174
átviteli sebesség 99-100
automatikus ismétlés kérés 145

B

batch processing (kötegelt feldolgozás) 16, 211, 215-216
baud 89
bázisregiszter 177-178, 223, 227, 237
bitparallel (párhuzamos) átvitel 101
bitsoros (soros) átvitel 102, 104
blokkolt program 183, 229, 233

C

ciklikus kód 143, 154
cimzés 222, 237, 239
cimzés, indirekt 239

CPU (központi vezérlő egység) 170
CPU idő 233
csatorna, I/O 172
csatorna, multiplexer 172
csatorna program 173
csatorna, szelektor 172
csatorna, zajos 135
csillaghálózat 114
csomóponti számitógép 200

D

data management (adatkezelés) 22, 37-40
DATEX hálózat 131
dekódolás 136
device management (eszköz kezelés) 40, 232-233
digitális átvitel, alapsávi 107
digitális átvitel, mintavételes 108
directory (file meghatározó mező) 38
duplex összekötelés 116,118,200
duplex vonal 101

E

erőforrás 18, 168, 174, 181,221
erőforrás megoszlás 89
eszközkezelés (device management) 40, 232-233

F

fázis moduláció 107
fejléc, üzenetben 149, 202-203
feldolgozási program 19, 20
fél-duplex összekötelés 116
fél-duplex vonal 101
fésüzés (interleaving) 174
file 37-38, 186-187, 241-243
file, direkt 38
file, index-szekvenciális 37
file kezelés 186-187, 241-243
file meghatározó mező (directory) 38
file, szekvenciális 37
file szervezés 37-38, 186
folyamat (process) 14, 181, 183-184, 219
folyamat irányitó rendszer 93
forrás, információ 134
forráskódolás 134
frekvencia modeláció 106

G

gépi architektura 68-69, 71-72, 168, 216-218

H

hagymahéj algoritmus 47
hálózat, komplex számitástechnikai 200-204
hálózat, többsiku 123
hálózat, üzenetkapcsolt 125-128, 149
hálózat, vonalkapcsolt 124-125
hálózatvezérlés 189
hibacsomó 142-143
hibafelismerés 136
hibajavitó kód 142
hibajelző kód 142
hibakorlátozás 136-146, 198
hibakorlátozási stratégia 137
hibakorlátozó kódolás 136-146
hiba, véletlen 142
hierarchikus tár 227-228
hozzáférési módszerek (access methods) 39

I

időjelző óra (time) 171, 233
időosztásos üzemmód (time sharing) 86, 93, 209, 255
időosztásos üzemmód definiciója 211-212
időosztásos üzemmód szolgáltatásai 213, 244-253

időszelet 87, 233
impulzus folyam átvitel 105-106
indirekt cimzés 239
inkrementális forditás 247-249
input-output rendszer 170, 171-174
input-output processzor 173
interaktiv forditás 245-247
interaktiv üzemmód 86-87, 244
interleaving (fésüzés) 174
interpretiv (forditás) 250
interrupt (megszakitás) 22-23, 170, 174-176, 188, 218, 233

J

jelszó 242
job 18
job befejezése 25
job control (munkavezérlő) nyelv 30, 36, 39
job management 21, 30-37
job step (job lépés) 18
job vezérlő blokk 25

K

katalógus 39, 186, 242
katalógus kezelés 40
késleltetett betöltés 53, 55
kiválasztás (selecting) 121, 150-151, 198
kód, arány 142
kód, ciklikus 143-144
kód, hibajavitó 142
kód, hibajelző 142
kód, hibakorlátozó 136-146
kódszó 134
kötegelt feldolgozás üzemmód (batch processing) 16, 211, 215-216
kötegelt távadatfeldolgozás (remote batch processing) 93
központi vezérlő egység (CPU) 170
központi vezérlő egység idő 233
közvetlen cimzés 69
külső tároló kezelése 40

L

lapkezelés (paging) 49-67, 178, 224-226
lapkezelési algoritmus 63-66, 183, 230-232
lapmutató 63
laptábla 63, 178, 224
lekérdezés (polling) 120-123, 148-154, 198
lekérdező rendszer 92

M

megbizhatóság 169
megszakitás (interrupt) 21-23, 170, 174-176, 188, 218, 233
megszakitás feldolgozása 26
megszakitás, gépi 175
megszakitás, input-output 175
megszakitás, külső 175
megszakitás, program 175
memória (operativ tár, központi tár) 170, 228
memória- (tár-) gazdálkodás 41-67, 69-72, 176-180, 184-185
memória- (tár-) védelem 176-180
modem 96, 163
moduláció 106-107
MULTICS 241
multiprogramozás 69, 233
munkavezérlő (job control) nyelv 30-36, 39

O

operációs rendszer stratégiája 16
operációs rendszer szerkezete 18
operlapping, tár elérésnél 174
overlay (átlapolás) 241
overlay, minimális 49-53

Ö

összeköttetés, felfüzéses 119-121, 148, 151, 154
összeköttetés, kétoldalu 118

összeköttetés, közvetlen (pont-pont) 118-119, 148, 151
összeköttetés, többoldalu 118

**P**

paging (lapkezelés) 49-67, 178, 224-226
paraméteres program 20
parancsnyelv 182
párhuzamos átvitel 101
paritásellenőrzés 142-143
partició 42, 44
partició, rögzitett nagyságu 42
partició, változó nagyságu 44
polling (lekérdezés) 120-123, 148, 150-154, 198
prioritás 21
privilégizált állapot 170
process (folyamat) 14, 181, 183-184, 219
processzor 173
processzor, adatátviteli 194-195
processzor, emulátor front-end 194-195
processzor, front-end 194-195
programcsere (swapping, roll-in, roll ont) 228-232
programgenerátor 20
puffer 187, 189, 198
pulzus kód moduláció 108

**R**

reál-idős (real time) rendszer 85, 212
redundancia 136
reentrant (ujrahivható, tiszta) program 177, 234
relokálhatóság 71, 177-178, 223
remote batch processing (kötegelt távadatfeldolgozás) 219
rezidens rutin 182
roll-in, roll ont (swapping, programcsere) 176, 228-232

**S**

scheduler (ütemező, tervező program) 26, 182-184
segédprogramok (utility) 21
selecting (kiválasztás) 121, 150-151, 198
software, megosztott 226
soros átvitel 102-105
spooling 233
start-stop (aszinkron) átvitel 102-104
supervisor (felügyelő program) 23-29, 41-42, 44
supervisor állapot 22
supervisor funkciók 25
swapping (programcsere, roll-in, roll-ont) 176, 228-232
system management 21, 23-27
szegmens 179
szegmens kapcsoló 240
szegmens leiró tábla 178, 237
szegmentálás 178-180, 235-241
szimplex összeköttetés 116
szimplex vonal 100
szinkron átvitel 104-105
szinkron átvitel, bit orientált 158-159
szinkron átvitel vezérlése 151-158
szinkronizálás 105
szöveg állapot 156

**T**

tár (memória) gazdálkodás 41-67, 69-72, 176-180, 184-185
tár (memória) védelem 176-180
tár modul, független 173
távadatfeldolgozás, off-line 85-86
távadatfeldolgozó hálózat 88-89
távadatfeldolgozó rendszerek osztályozása 92-94
távolsági kötegelt feldolgozás (remote batch processing) 93
terminál 86, 96
terminál csatlakoztatása 160-164
terminál, intelligens 162, 166, 196-197

terminál, pufferelt 166
terminál vezérlő egység 96, 161-162, 166
tervező (ütemező, scheduler) program 26, 182-184
teszt-utasitás, dinamikus 252
time-sharing (időosztásos) üzemmód 86, 93
timer (időjelző óra) 171, 233
tiszta (ujrahivható, reentrant) program 177, 234

**U**

utility (segédprogram) 21
ütemező (tervező, scheduler) program 26, 182-184
üzenet 98
üzenet csomag 200-202
üzenet kapcsolás 95, 113, 125, 200-202
üzenetkapcsoló rendszer 93
üzenet sorbaállitás 189
üzenet szerkesztés 191, 198
üzenet visszacsatolás 144

**V**

várakozási sor 184, 188, 189
vegyes hálózat 116
vegyes hálózat, több pontos 119
vétel 154, 156
vezérállomás 118, 147
vezérlési állapot 156
vezérlő program 19, 21, 41
virtuális cim 124-125, 135
virtuális processzor 87, 169, 220
virtuális tár 126-128, 220, 226-227, 230-232
vonal-csatlakozó 96, 161-163, 167
vonal kapcsolás 95, 124-125
vonal-koncentrátoros hálózat 112-113

# Idegen nyelvű összefoglalók

Thomas Bakos

## OPERATING SYSTEMS

The software industry - that has been created together with the appearance of the third-generation computers - is a product of the dynamically developing computer science and computing technique. When studying software products, the operating systems are to be found in the first place. These systems give a base for the majority of other software products.

The study makes a survey on operating systems working in batch-processing. The survey is more detailed as it is needed for an average user, moreover in some respects it is made for those of working in the field of program development.

Corresponding to this, the first chapter starts out discussing the definition of the operating systems after the introductory part. The second chapter is containing the general structure of the operating systems. Storage management is selected to illustrate various resource management functions of the operating system in detail. Chapter 3 shows several storage allocation methods. This chapter, together with the fourth one - that discusses the interrelation of operating systems and machine structure - was made primarily for system analysts and programers, but it can be useful for all users to make them understand the functions of operating systems.

Miklós Lohonyai

## TELEPROCESSING

In our days teleprocessing is considered to be a very important method of computing techniques and its importance is to increase steadily in the forthcoming years.

This study covers some problems characteristic to teleprocessing systems with a special care to those of related to data transfer. In a general review of technical support it starts out discussing the characteristics and special features of telecommunication lines and telecommunication networks, different coding and error-eliminating processes, the control of telecommunication, the hardware of the terminals and that of the central computer and some special software problems originating from teleprocessing.

In addition to the above summary the author exposes clear-out directives for the creation of a teleprocessing terminology.

Zoltán Huba

## TIME-SHARING SYSTEMS

This study deals with all those requirements and circumstances that created an urgent need for the realization of the time-sharing operation (TSO). It gives definitions for the most important characteristics of the time-sharing systems (TSS), and informs of their various forms of realization.

After having made clear the terminology it details some economic efficiency questions considering the introduction of the TSO. It lets the reader know the basic functions of the TSS control software, the economic use of main storage, the dynamic allocation of resources and file-handling. The author gives an extra case-study for demonstrating the use of the conversational operation mode of the TSS, which mode has an outstanding importance for the programers.

Тамаш Бакош

## ОПЕРАЦИОННЫЕ СИСТЕМЫ

Производство программного обеспечения - которое создалось вместе с ЭВМ третьего поколения - является продуктом быстрого развития науки и протышленности вычислительной техники.

В ходе изучения продуктов программного обеспечания на первом месте находятся операционные системы, которые большей частью служат основой других продуктов программного обеспечения.

Статья занимается операционными системами, работающими в режиме пакетной обработки. В некоторых аспектах изложение проблемы является более подробным, чем это требуется для массового использования, оно касается уже занимающихся, развитием системы.

Соответственно этому I. глава занимается - кроме ввевения - определением операционных систем, 2. глава содержит описание общей структуры операционных систем, 3. глава заключает в себе - из средств программирования, использованных в операционных системах - разные методы использования запоминающего устройства и вместе с 4. главой, которая освещает взаимодействие операционных систем и машинной структуры, касается в первую очередь проектировщиков и программистов, но помогает также всем использователем в осознании функции операционных систем.

Миклош Лохонаи

## ОБРАБОТКА ДАННЫХ НА РАССТОЯНИЕ

Обработка данных на расстояние является уже в настоящее время очень важным методом вычислительной техники и в недалеком будущем ее значительность будет возрастать еще лучше.

Стать занимается вопросами, характеризующими системы обработки данных на расстояние, в особенности связаннвми с передачей данных на расстояние. В рамках общегс изложения системной техники в статье рассматриваются свойства и характерные черты каналов и сетей передачи данных, методы шифровки и защиты ошибок, управление передачей данных, техническое обеспечение терминалов и центральной ЗВМ и специальные проблемы программиного обеспечения в связи с обработкой данных на расстояние.

Кроме краткого изложения системной техники автор однозначно указывает на разработку технических терминологий по обработке данных на расстояние.

Золтан Хуба

## СИСТЕМЫ РАЗДЕЛЕНИЯ ВРЕМЕНИ

Статья излагает требования и обстоятельства, вследствие которых стало необходимым осуществить режим разделения времени. Она определяет главные характерные черты системы и сообщает о разных фортах ее осуществления. После выяснения терминологии статья занимается вопросами рентабельности создания режима. Она излагает основные йункции программного обеспечения для Управления системами разделения времени, использование запоминающего устройства, динамическое расиределение устройсть, обращение с файлами.

Автор представляет использование ЭВМ в режиме диалога, который является для программистов самым важным при использовании систем разделения времени.

A kiadásért felel: Kecskés József igazgató
Kiadói szerkesztő: Hoványi János
Müszaki szerkesztő: Sebestyén László
Boritóterv: Bogár Imre
A kézirat nyomdába érkezett: 1974. VI. 15.
Imprimálva: 1974. XI. 30.
Megjelent 1974. XII. 20.
Formátum: B/5
Terjedelem: 23,45 (A/5) iv
Példányszám: 3 000
Statisztikai Kiadó Vállalat, Nyomdaüzem - 128574

82,– Ft

R10
KISSZÁMÍTÓGÉP

V állalatirányítás
I ntegrált adatfeldolgozás
D D C
E gyetemi oktatás
O n-line adatgyűjtés
T ávadatfeldolgozás
O ff-line adatelőkészítés
N umerikus szerszámgép vezérlés

Részletes tájékoztatást nyújt:

VT
TV VIDEOTON
SZÁMÍTÁSTECHNIKAI GYÁRA

1021 Budapest,
Vöröshadsereg útja 54.
Telefon: 213-187

A számítástechnika legújabb eredményei

1